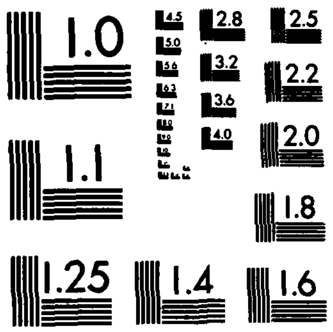
AD-A123 566 INTERFACE SPECIFICATIONS FOR THE SCR (A-7E) EXTENDED COMPUTER MODULE(U) NAVAL RESEARCH LAB WASHINGTON DC D L PARNAS ET AL. 06 JAN 83 NRL-MR-4843 1/2

UNCLASSIFIED F/G 9/2 NL


1.0
4.5
5.0
5.6
6.3
7.1
8.0
9.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

2

NRL Memorandum Report 4843

Interface Specifications for the SCR (A-7E) Extended Computer Module

D. L. PARNAS,* D. M. WEISS, AND P. C. CLEMENTS

Computer Science and Systems Branch
Information Technology Division

**University of Victoria*
Victoria, B.C.

K. H. BRITTON

IBM
Research Triangle Park, NC 27709

January 6, 1983

NAVAL RESEARCH LABORATORY
Washington, D.C.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NRL Memorandum Report 4843	2. GOVT ACCESSION NO. A123 566	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) INTERFACE SPECIFICATIONS FOR THE SCR (A-7E) EXTENDED COMPUTER MODULE		5. TYPE OF REPORT & PERIOD COVERED Interim report on a continuing NRL problem. 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) D.L. Parnas,* D.M. Weiss, P.C. Clements, and K.H. Britton†		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Research Laboratory Washington, DC 20375		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62712N; XF21242101; 75-0106-02
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Research Laboratory Washington, DC 20375		12. REPORT DATE January 6, 1983 13. NUMBER OF PAGES 116
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

*Present address: University of Victoria, Victoria, B.C.
†Present address: IBM, Research Triangle Park, NC 27709

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Abstract interfaces, Avionics software, Information hiding, Modular decomposition, Modules, Real-time systems, Software engineering, Software maintenance, Software specifications

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This document describes the programmer interface to a computing machine partially implemented in software. The Extended Computer is part of NRL's Software Cost Reduction (SCR) project, to demonstrate the feasibility of applying advanced software engineering techniques to complex real-time systems in order to simplify maintenance. The Extended Computer allows code portability among avionics computers by providing extensible addressing, uniform i/o and data access, representation-independent data types, uniform event signalling, a standard subprogram invocation mechanism, and parallel

(Continues)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. ABSTRACT (Continued)

process capability. The purpose of the Extended Computer is to allow the remainder of the software to remain unchanged when the host computer is changed or replaced.

This report describes the modular structure of the Extended Computer, and contains the abstract interface specifications for all the facilities provided to users. It serves as development and maintenance documentation for the SCR software design, and is also intended as a model for other people interested in applying the abstract interface approach on other software projects.

CONTENTS

INTERFACE SPECIFICATIONS FOR THE SCR (A-7E) EXTENDED COMPUTER MODULE

INTRODUCTION

The Extended Computer (EC) is a computing machine partially implemented in software. It was designed as part of the Software Cost Reduction (SCR) project at the Naval Research Laboratory. The design goals are 1) code portability, 2) abstraction from computer hardware idiosyncracies, 3) more easily understood code, and 4) sharing of solutions to common machine dependent coding problems. The Extended Computer is designed to be efficiently implemented on avionics computers such as the IBM 4PI TC-2. The instruction set allows straightforward, efficient code generation using a macroprocessor.

The Extended Computer has the following features:

1) Extensible addressing: There is no syntactic limit to the amount of memory that can be addressed. The actual memory size is a parameter that is set at system-generation time.

2) Uniform data access: Hardware addressing techniques, such as use of base and link registers, are hidden from programmers.

3) Uniform subprogram access: Macro calls and subroutine calls appear identical in the calling program.

4) Uniform input/output: Variations in I/O operations are hidden. All input (output) data items are read (written) using the same instructions.

5) Uniform event signalling: The difference between hardware interrupts and software-detected events is hidden. All interrupt handling is hidden.

6) Data types: Data types representing reals, bitstrings, and time intervals are provided together with the necessary conversion functions. Data representations are hidden. Hardware arithmetic and bitstring operations are hidden.

7) Parallel processes: Programs can be written as a set of cooperating sequential processes. The number of hardware processors and their scheduling are hidden.

8) State control: Computer state transitions among various states (including off, operating, and failed) are signalled to the user programs. The mechanics of state transitions are hidden.

9) Built-in test: Diagnostic programs to test the integrity of memory and the correct operation of the hardware are built-in. The tests and evaluation criteria are hidden.

Manuscript approved April 30, 1982.

10) Exception handling: Both a development version, with extensive checks for programming errors, and a production version are available. Programs that cause no undesired events [WUER76] on the development version will compute the same values on both versions. The version can be selected at system-generation time.

11) Efficient programs without the go to: Instead of a program counter and commands that cause branches, the Extended Computer includes the it ti control structure described in [ITTI1] and [ITTI2].

The Extended Computer has been designed to hide the interface characteristics of a computer with capabilities similar to those of the IBM 4PI/TC-2. Were the present A-7 computer to be replaced by one with different capabilities, we would shift some responsibilities to/from other parts of the software. For example, if the new computer used an external device for timing, the implementation of the timeint data type would become a part of the device interface modules. Or, if the new computer included a capability for angle implementation, the machine-independent implementation of an angle data type would be replaced by a machine-dependent module that was part of the EC, but with the same interface as the present angle data type. Of course, under such unlikely circumstances, the appropriate documentation (such as [REQ], [MG], and [AT], as well as this document) would be changed to remain consistent with the new hardware. If the EC design were to be used in an application that did not require all of its capabilities, a compatible subset could be used.

We recommend that this procedure be followed by anyone maintaining this system, and by those who are designing other systems using a similar approach.

This document specifies the user interface to the Extended Computer. The contents, form, and notation are in accordance with the guidelines given in [SO], with the following additions.

Subsection 2: Interface Overview

In addition to the contents described in [SO], the Interface Overview subsection of each specification may contain the following entries:

Instruction Templates: In contrast to access programs, EC instructions take programs (lists of executable EC statements) as parameters. The Instruction Template table describes the format of instructions provided by the module. Each instruction begins with a keyword suggesting the instruction's purpose and usually ends with that keyword reversed. Additional keywords are each followed by a parameter list. The parameters are numbered separately for each keyword. Instruction Template tables use the same parameter notation used in the Access Program table.

Instructions follow the same rules as access program calls, except that parameters in an instruction are not enclosed in parentheses. Instructions bracketed by "++" may only be invoked at system-generation time.

Built-in Objects: The built-in object table lists special keywords representing objects provided by the module.

Subsection 3.2: Undesired Event Assumptions

The only undesired event assumptions documented in this section will be those concerning undesired events that cannot be detected until run-time.

EC.DATA
DATA MANIPULATION FACILITIES

EC.DATA.1 INTRODUCTION

EC.DATA.1.1 ENTITIES

The Extended Computer provides literals, constants, and variables. We refer to these as entities. Literals are values appearing in programs. Constants have names and values; run-time programs can read the values but not change them. Variables have names and values; the values can be read or written by run-time programs. A register is a variable with a faster access time than other variables. There is one register for each process (see EC.PAR). All constants and variables other than registers may be accessed from any process of the program. It is possible to declare arrays of variables or constants. An element of an array may be used as an individually declared entity of the same type. Users are given the facility for providing information to the Extended Computer about the relative speeds with which declared entities should be accessed.

EC.DATA.1.2 TYPES

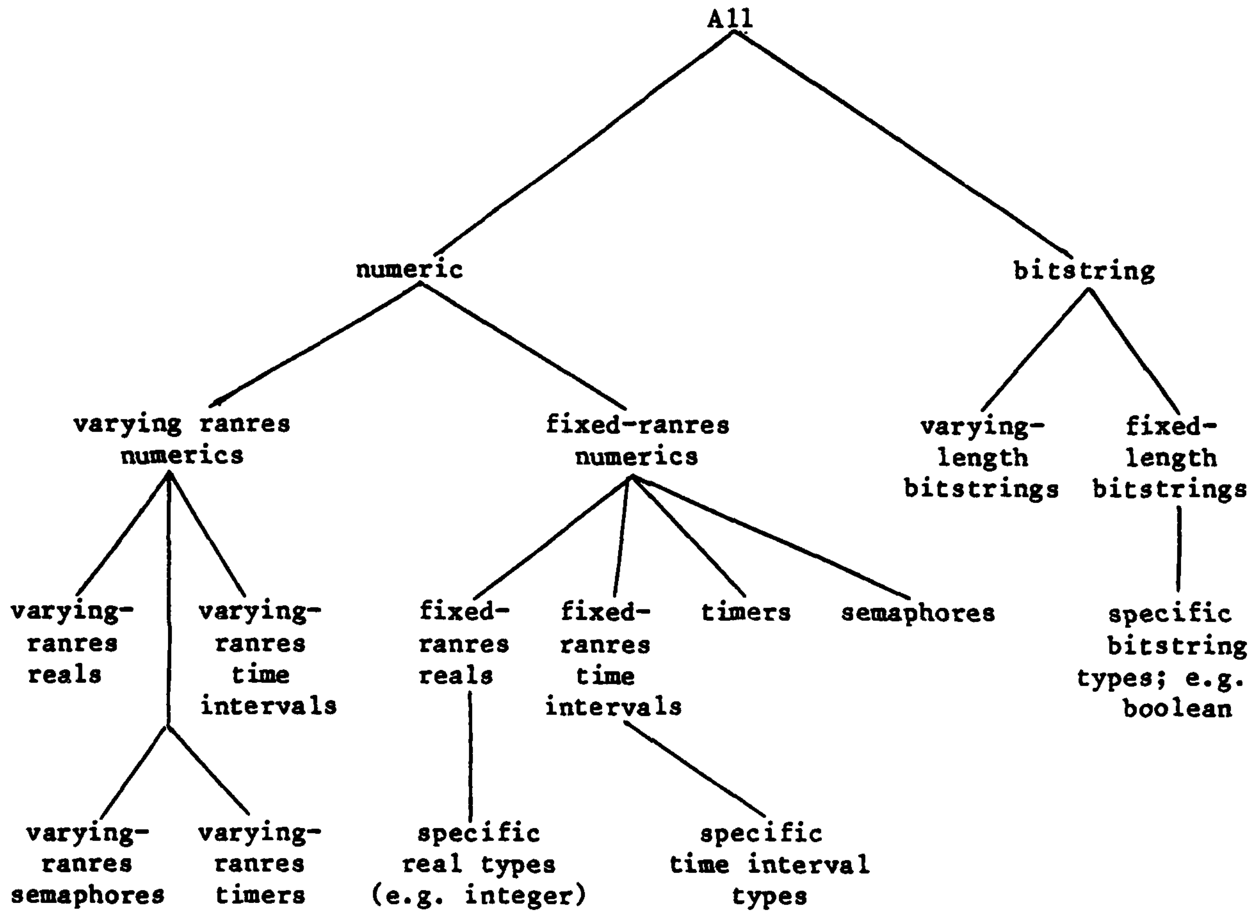
Types are classes of entities. The Extended Computer provides a hierarchy of types; at the highest level are numeric and bitstring. Numeric types are characterized by range and resolution; bitstring types are characterized by length. The value of a characteristic for an entity is called an attribute.

A type class is a type that contains entities with different behavior. A specific type (also called "spectype") is a subclass of a type class in which all variables have identical behavior, i.e., they can take on the same set of values and one may perform the same operations on them. The behavior of the program will not change if two variables of the same specific type are interchanged throughout the program.

For each type class, there are any number of specific types with fixed (but different) attributes. There is also a specific type whose attributes can vary at run-time.

Figure 1 provides an overview of the EC data types by showing the Extended Computer's type classes and specific types. Lines connect a type with its sub-types. The terminal nodes represent specific types.

The Extended Computer provides two numeric type classes illustrated in Figure 1, but not described in this chapter. They are semaphores and timers, whose operations are described in EC.PAR.3 and EC.SEQ.3, respectively.


All
numeric
bitstring
varying ranres numerics
fixed-ranres numerics
varying-length bitstrings
fixed-length bitstrings
varying-ranres reals
varying-ranres time intervals
fixed-ranres reals
fixed-ranres time intervals
timers
semaphores
specific bitstring types; e.g. boolean
varying-ranres semaphores
varying-ranres timers
specific real types (e.g. integer)
specific time interval types

Not all of these types are currently implemented; to see which ones, refer to Appendix 4.

Figure 1

EC.DATA.1.3 SCALAR LITERALS

A scalar literal belongs to a particular type and may belong to more than one specific type; acceptable formats for scalar literals are defined in the table below.

In the format descriptions,

(A, B, C, ...)^n means a sequence of length n composed of elements selected from the list of elements within the parentheses.

Table EC.DATA.a: Literal Formats

Type	Values
boolean	\$true\$ OR \$false\$; alternatively 1 OR 0
real	numeric literal, in one of the following formats: standard decimal notation e.g 112.345, .000234, 127 exponent notation: decimal number, followed by :En, where n is an integer, meaning that the value is the number multiplied by 10^n e.g. 1.12345E2 (= 112.345); 2.34E-4 (=.000234) power of two notation 2**n, e.g., 2**3 (for 8)
bitstring	bitstring literal: (0, 1)^n :B, where n is a positive integer, e.g. 011101:B
timeint	There are no literals of type timeint. Values of type timeint can be specified by using real literals and one of the conversion programs provided to convert real values to timeint values. p2 in the conversion program is not supplied when used in declarations or to provide literals.

Array literals are described in EC.DATA.4 under "arraylit".

EC.DATA.1.4 REGISTERS

A single register is provided for each process. Operations using the register are likely to be faster than operations on other variables. A register is a variable with varying type-class and varying attributes; each instruction that uses a register must include information sufficient to determine a type class and the appropriate attributes.

A process cannot access the register of another process.

A parameter that is specified as the !+destination+! of an access program defined in this chapter may be a register only, or a list of variables possibly including the register.

The contents of a register may be changed by a) using the ister as a destination, or b) performing an operation without specifying t the register contents be preserved. Each access program defined his chapter may appear with or without a suffix "-SAVE" (e.g. +MINUS+ or S-SAVE+). Use of the suffix specifies that the contents of the register [illegible] be preserved by the operation. Omission of the suffix indicates that the register contents need not be preserved.

Table EC.DATA.b shows how the value of a register is affected by an operation. Value undefined indicates that the value contained in the register is unspecified. If a program reads a register when its value is undefined, the results will be unpredictable.

Table EC.DATA.b: Effects of Operations on Register Contents

Register use	Suffix	Effect of the operation on Register
read only	none	value undefined
read only	-SAVE	value not changed
written or read and written	none	new value produced by operation
written or read and written	-SAVE	undesired event
not referenced	none	value undefined
not referenced	-SAVE	value not changed

EC.DATA.2 INTERFACE OVERVIEW

EC.DATA.2.1 DECLARATION OF SPECIFIC TYPES

All specific types must be declared and given a name. Numeric types are characterized by range and resolution, bitstring types by length. The type declaration must indicate whether or not these attributes can vary at run-time. For some types, the EC allows users to choose among different versions of the implementation; each version is especially efficient for performing certain operations. The versions, and the advantages/disadvantages of each, are specified in Appendix 6.

Program name	Parm type	Parm info	Undesired events
++DCL_TYPE++	p1:name;I	name of new type	%name in use%
	p2:typeclass;I	containing typeclass	%inappropriate attributes%
	p3:attribute;I	attributes of type	
	p4:binding;I	Can attributes vary at run-time?	
	p5:version;I_OPT	implementation version	

Program Effects

A specific type that is a member of type class p2 and has attribute p3 with binding p4 and implementation version p5 is declared to have identifier p1. The identifier can be used as the spectype (p2) parameter in calls on ++DCL_ENTITY++ and ++DCL_ARRAY++ in programs that follow the declaration.

EC.DATA.2.2 DATA DECLARATIONS

EC.DATA.2.2.1 DECLARATION OF VARIABLES AND CONSTANTS

Variables and constants must be declared before they are used. The declaration must specify the name of the new entity, a previously declared specific type (one of the terminal nodes on the tree of figure 1), whether the entity is a constant or a variable, and an initial value.

Program name	Parm type	Parm info	Undesired events
++DCL_ENTITY++	p1:name;I	entity name	%undeclared spectype%
	p2:spectype;I	specific type of entity	%inappropriate% attributes%
	p3:attribute;I_OPT	initial attribute	%name duplication%
	p4:convar;I	when writeable?	%unknown value%
	p5:constant or literal whose value is in domain of type named by p2;I	initial value	%wrong init value type%

Program Effects

An entity with identifier p1, spectype p2, initial attribute p3, and initial value p5 is declared. p3 is supplied if the spectype p2 has varying attributes. The entities that have been declared may be used as operands in the programs that follow.

EC.DATA.2.2.2 DECLARATION OF ARRAYS

Program name	Parm type	Parm info	Undesired events
++DCL_ARRAY++	p1:name;I	array name	as above, plus...
	p2:spectype;I	element type	
	p3:attribute;I_OPT	initial attribute	%wrong init value size%
	p4:convar;I	when writeable?	
	p5:sequence of constants and/or arraylits whose values are in domain of type named by p2;I	initial value	%illegal index set%
	p6:indexset;I	array indices	

Parameters

The set of parameters p2 through p6 can be repeated any number of times, allowing arrays of different specific types. The union of the named index sets must result in a contiguous set of integers.

Program Effects

A one-dimensional array with identifier p1, spectype p2, initial attribute p3, initial value p5, and index set p6 is declared. p3 is supplied if the spectype p2 has varying attributes. Elements of the array can be used wherever an entity of the same specific type could be used. An array may also be a parameter to a user-defined program.

EC.DATA.2.3 ACCESS SPEED RANKING OF DATA

The Extended Computer can implement a "not-slower-than" relation between any two variables, constants, or arrays.

Program name	Parm type	Parm info	Undesired events
++RANK_DATA++	p1:name;I	entity or array name	%undeclared operand%
	p2:name;I	entity or array name	

Parameters

p1 and p2 each must be one of:
- the name of a previously-declared constant or variable;
- the name of a previously-declared array.

Users may supply any number of pairs of parameters p1 and p2 in the same call.

Program effects

For each given pair (p1,p2), using p1 as an operand in an EC program will take no longer than using p2 in the same program.

EC.DATA.2.4 OPERAND DESCRIPTIONS

The data manipulation programs defined in this chapter take entities or lists of entities as operands. Table EC.DATA.c describes the form of the description of individual operands. The parentheses shown are required.

Table EC.DATA.c INDIVIDUAL OPERAND SPECIFICATION

Nature of Operand	Form of Operand
Literal	Literal value
Constant or variable with fixed attributes	Name of entity
Variable with varying attributes	(Name of operand, information determining attributes)
Array element	(array name, array index, information that would be given for an individual entity of the same specific type) Array index must be an integer.
Register	(REG, information determining typeclass and attributes)

Information determining typeclass and attributes: The information may be either the name of a specific type with fixed attributes, or the attributes themselves. Attributes may be given as literals, variables, or constants; see explanation of "attribute" in EC.DATA.4 for further explanation.

Inconsistent information: If the information given when a variable with varying attributes is used as a source (i.e., input parameter) is not the same as that given when that variable was most recently used as a destination (i.e., output parameter), the results are undefined.

Multiple operand destinations: Any operand described as a !+destination+! may also be a parenthesized list of destinations. All of the destinations will receive the same value; the assignments will be made in an arbitrary order, or simultaneously.

Repeated operand lists: In any operation, the list of operands may be repeated as often as desired. Such a construct describes a set of operations, each element of the set corresponding to one of the operand lists; the operations may be performed in an unspecified order or simultaneously.

<u>Undesired Events</u>: The following Undesired Events can occur when operands are specified:

%constant destination%
%illegal array index%
%inappropriate attributes%
%inconsistent register access%
%undeclared operand%
%undeclared spectype%

EC.DATA.2.5 TRANSFER OPERATIONS

Program name	Parm type	Parm info	Undesired events
+SET+	p1:see below;I	!+source+!	%inconsistent lengths%
	p2:see below;O	!+destination+!	%range exceeded%
			%list mismatch%

Parameters

p1 and p2 are parenthesized lists of operands separated by commas. The lists must both have the same length. If the lists only have one element, the parentheses may be omitted. Corresponding elements of p1 and p2 must be either both real, or both timeint, or both bitstrings of the same length. Extensions of +SET+ to allow additional operand types will be introduced in later chapters.

Program Effects

+SET+

Let $p1_i$ denote the i^{th} element of p1, and $p2_i$ denote the i^{th} element of p2. For all i, $p2_i$ = the value of $p1_i$ before the execution of +SET+.

Note that if p1 and p2 are disjoint, the assignments may be done in any order. If p1 and p2 have elements in common, the assignment will have the same effect as if all values of p2 were saved in temporaries and then the assignment was made to p1. This is equivalent to Dijkstra's concurrent assignment statement [DIJK77].

EC.DATA.2.6 NUMERIC OPERATIONS

EC.DATA.2.6.1 NUMERIC COMPARISON OPERATIONS

Program name	Parm type	Parm info	Undesired events
+EQ+	p1:see below;I	!+source+!	None
+NEQ+	p2:see below;I	!+source+!	
+GT+	p3:boolean;O	!+destination+!	
+GEQ+	p4:see below;I_OPT	!+user threshold+!	
+LT+			
+LEQ+			

Parameters

p1,p2,p4 must be either all real types or all timeint types.

Program Effects

+EQ+	p3 = (p1 = p2)*
+NEQ+	p3 = NOT (p1 = p2)*
+GT+	p3 = p1 - p2 is positive and NOT (p1 = p2)*
+GEQ+	p3 = (p1 = p2)* OR (p1 - p2 is positive)
+LT+	p3 = p1 - p2 is negative and NOT (p1 = p2)*
+LEQ+	p3 = (p1 = p2)* OR (p1 - p2 is negative)

*Definition of equality (=):

absolute value(p1 - p2) is less than or equal to threshold, where threshold is either p4 or, if p4 is not supplied, one-half of the larger of resolution (p1) and resolution (p2).

Extensions of +EQ and NEQ+ to allow additional operand types will be discussed in later chapters.

EC.DATA.2.6.2 NUMERIC CALCULATIONS

Program name	Parm type	Parm info	Undesired events
+ABSV+	p1:see below;I	!+source+!	%range exceeded%
+COMPLE+	p2:see below;O	!+destination+!	
+ADD+	p1:see below;I	!+source+!	
+MUL+	p2:see below;I	!+source+!	
+SUB+	p3:see below;O	!+destination+!	
+DIV+	p1:see below;I	!+source+!	%range exceeded%
	p2:see below;I	!+source+!	%divide by zero%
	p3:see below;O	!+destination+!	
	p4:same as p3;I OPT	!+max div result+!	
	p5:same as p3;I_OPT	!+fall back value+!	

Parameters

+ADD+
+ABSV+
+COMPLE+
+SUB+

(1) all operands real, or
(2) all operands timeint

+MUL+

(1) all operands real, or
(2) one of p1 or p2 real, the other operands timeint

+DIV+

(1) p1,p2 and p3 real, or
(2) p1 and p2 timeint and p3 real, or
(3) p1 timeint, p2 real, p3 timeint;

Program Effects

```
+ABSV+        p2 = magnitude(p1)
+ADD+         p3 = p1 + p2
+COMPLE+      p2 = - p1
+MUL+         p3 = p1 * p2
+SUB+         p3 = p1 - p2
+DIV+         (   (division successful*)
                        then  p3 = p1/p2
                  (p5 supplied AND division unsuccessful*)
                        then p3 = p5 * SIGN(p1/p2)
                  (p5 omitted AND division unsuccessful*)
                        then p3 undefined    )
```

*+DIV+ can sometimes result in the loss of all significance; when this happens, it is said to be unsuccessful. Division will not be successful if p4 (if supplied) is less than p1/p2, but will be successful if p4 gteq p1/p2, or p4 is omitted. If p4 is omitted, +DIV+ will be slower than if p4 is provided.

Table EC.DATA.d

Resolution of Numeric Calculation Results

OPERATION	RESOLUTION OF RESULT
+ADD+(p1,p2,result) +SUB+(p1,p2,result)	MAX(resolution(p1), resolution(p2))
+MUL+(p1,p2,result)	resolution(p1) * resolution(p2)
+DIV+(p1,p2,result,p4,p5)	unsuccessful and p5 given: resolution(p5) otherwise, resolution(p1) / resolution(p2)
+ABSV+(p1,result) +COMPLE+(p1,result)	resolution(p1)
Conversion between real and another numeric typeclass	Conversion of the resolution of the source

EC.DATA.2.6.3 OPERATIONS CONVERTING OTHER TYPES TO REALS

Program name	Parm type	Parm info	Undesired events
+R_BITS_2COMP+ +R_BITS_POSITIVE+ +R_BITS_SIGNMAG+	p1:bitstring;I p2:integer;I p3:real;O	!+source+! !+radix pt ident+! !+destination+!	%range exceeded%
+R_TIME_HOUR+ +R_TIME_MIN+ +R_TIME_MS+ +R_TIME_SEC+	p1:timeint;I p2:real;O	!+source+! !+destination+!	

Program effects

+R_BITS_2COMP+	p3 = real value equivalent to p1 assuming that bitstring p1 is in a two's complement representation, bit 0 is the most significant bit, and the radix point is specified by p2.
+R_BITS_POSITIVE+	p3 = real value equivalent to p1 assuming that bitstring p1 represents a positive number, with bit 0 the most significant bit, and the radix point is specified by p2.
+R_BITS_SIGNMAG+	p3 = real value equivalent to p1 assuming that bitstring p1 is in a sign magnitude representation, bit 0 is the most significant bit, and the radix point is specified by p2.
+R_TIME_HOUR+	p2 = a real value giving the time p1 in hours.
+R_TIME_MIN+	p2 = a real value giving the time p1 in minutes.
+R_TIME_MS+	p2 = a real value giving the time p1 in milliseconds.
+R_TIME_SEC+	p2 = a real value giving the time p1 in seconds.

EC.DATA.2.6.4 OPERATIONS CONVERTING TO TIME INTERVALS

Program name	Parm type	Parm info	Undesired events
+T_REAL_MS+	p1:real;I	!+source+!	%range exceeded%
+T_REAL_SEC+	p2:timeint;O	!+destination+!	
+T_REAL_MIN+			
+T_REAL_HOUR+			

Parameters

If p1 is given as a literal, p2 may be omitted and the call used as a literal of type timeint. If p1 is given as a constant, p2 may be omitted and the call used anywhere that a constant of type timeint may be used.

Program effects

+T_REAL_MS+	p2=timeint value equivalent to p1 assuming p1 to specify the time interval in milliseconds.
+T_REAL_SEC+	p2=timeint value equivalent to p1 assuming p1 to specify the time interval in seconds.
+T_REAL_MIN+	p2=timeint value equivalent to p1 assuming p1 to specify the time interval in minutes.
+T_REAL_HOUR+	p2=timeint value equivalent to p1 assuming p1 to specify the time interval in hours.

EC.DATA.2.7 OPERATIONS FOR THE BITSTRING TYPE CLASS

Bits in all bitstring types are numbered from 0 upward. We refer to bit 0 as the leftmost bit and a shift of information from higher numbered bits to lower numbered bits as a left shift.

EC.DATA.2.7.1 BITSTRING COMPARISON OPERATIONS

Program name	Parm type	Parm info	Undesired events
+EQ+	p1:bitstring;I	!+source+!	%inconsistent lengths%
+NEQ+	p2:bitstring;I	!+source+!	
	p3:boolean;O	!+destination+!	

Program Effects

```
+EQ+          p3 = (p1 = p2)*
+NEQ+         p3 = NOT (p1 = p2)*

*Definition of equal (=)     length(p1) = length(p2) and
                             for all i such that 0 lseq i lt length(p1)
                             bit (i) of p1 = bit (i) of p2
```

EC.DATA 2.7.2. BITSTRING CALCULATIONS

Program name	Parm type	Parm info	Undesired events
+AND+	p1:bitstring;I	!+source+!	%inconsistent lengths%
+CAT+	p2:bitstring;I	!+source+!	
+MINUS+	p3:bitstring;O	!+destination+!	
+NAND+			
+OR+			
+XOR+			
+NOT+	p1:bitstring;I	!+source+!	%inconsistent lengths%
	p2:bitstring;O	!+destination+!	
+REPLC+	p1:bitstring;I	!+source+!	%nonexistent position%
	p2:integer;I	source start position	
	p3:integer;I	destination start position	
	p4:integer;I	length	
	p5:bitstring;O	!+destination+!	
+SHIFT+	p1:bitstring;I	!+source+!	%inconsistent lengths%
	p2:integer;I	shift length	
	p3:direction;I	shift right or left	
	p4:bitstring;O	!+destination+!	

Parameters

+SHIFT+	p1 must be a parenthesized list of bitstring entities separated by commas. If there is only one element, the parentheses may be omitted.

Program Effects

+AND+	p3 = p1 AND p2
+CAT+	p3 = p1 followed by p2
+MINUS+	p3 = p1 AND (NOT p2)
+NAND+	p3 = NOT (p1 AND p2)
+NOT+	p2 = NOT p1
+OR+	p3 = p1 OR p2
+REPLC+	p5[p3:p3+p4-1] = p1[p2:p2+p4-1]
+SHIFT+	p4 = shift of the catenation of the elements of list p1, by p2 positions in the p3 direction. The vacated bits are set to "0".
+XOR+	p3 = (p1 AND (NOT p2)) OR (p2 AND (NOT p1))

EC.DATA.2.7.3. OPERATIONS CONVERTING TO BITSTRING

Program name	Parm type	Parm info	Undesired events
+B_REAL_2COMP+	p1:real;I	!+source+!	%left truncation%
+B_REAL_POSITIVE+	p2:integer;I	!+radix pt ident+!	
+B_REAL_SIGNMAG+	p3:bitstring;O	!+destination+!	

Program Effects

+B_REAL_2COMP+	p3 = two's complement representation of p1, such that the radix point of the resulting bitstring is positioned according to p2. Bit 0 of p3 will be the most significant. The operation truncates all bits beyond the highest numbered bit in the destination bitstring.
+B_REAL_POSITIVE+	p3 = bitstring representation of ABSV(p1), such that the radix point of the resulting bitstring is positioned according to p2. Bit 0 of p3 will be the most significant bit. The operation truncates all bits beyond the highest numbered bit in the destination bitstring.
+B_REAL_SIGNMAG+	p3 = sign magnitude representation of p1, such that the radix point of the resulting bitstring is positioned according to p2. Bit 0 will be the sign bit and bit 1 the most significant bit of the magnitude. The operation truncates all bits beyond the highest numbered bit in the destination bitstring.

EC.DATA.3 Undesired Event Assumptions

1. User programs will not divide by zero.

2. The result of any operation will not be outside the range of the destination variable.

3. In a replace operation, user programs will not specify positions that do not appear within bitstrings or specify a substring with a start position that is higher than the stop position.

4. After converting a numeric value to a bitstring, there will be no bits to the left of the most significant bit of the destination bitstring.

5. Users will not supply a parameter in an array reference that is not in the index set of the array.

EC.DATA.4 Local Type Definitions

arraylit — A set of literal values for a contiguous set of elements of an array. The values must be of the same typeclass as the array. The value for an element must also fall within the range of the specific type declared for that element. The syntax is:

(value sequence)	where value sequence is a sequence of literals or arraylits separated by commas; or,
(repetition factor*(value sequence))	where repetition factor is a positive integer

<u>Examples:</u>

(10, 2, -6)	Array of size 3, first element 10, second 2, third -6.
(15*(3.14))	size 15: all elements are 3.14.
(3*(5, 2*(0), 1))	size 12: (5,0,0,1,5,0,0,1,5,0,0,1)

attribute	Either 1) a positive integer specifying length for bitstrings or 2) an ordered triple of numeric entities specifying lower bound, upper bound and resolution for numerics. The lower bound and upper bound are often refered to collectively as the range attribute.
binding	Either FIX (meaning attributes do not change at run time) or VARY (meaning attributes may change at run time)
bitstring	An ordered list of values, each value represented by "0" or "1". The number of such values is called the <u>length</u> of the bitstring.
boolean	Bitstring of length 1. Where convenient, \$true\$ may denote "1:B", \$false\$ may denote "0:B".
convar	Either ASCON (meaning constant that will not change without a reassembly) or LOADCON (meaning constant that may be changed by a memory loading device while the program is not running) or VAR (meaning variable).
direction	Either R (meaning to the right) or L (meaning to the left).
indexset	A set of permissible indices. Only sets of contiguous integers may be created. The set must be specified in the following way: (si..li) where "si" denotes the smallest index and "li" denotes the largest index. Both si and li must be integers. For example, (7..12) indicates a six-element array indexed by the integers from 7 through 12. (-4..-4) indicates a one-element array who index is -4.
integer	Real with resolution = 1.
name	An identifier for an object created. A name must begin with an alphabetic character and consist only of alphanumerics.
real	An approximation to conventional real numbers.
spectype	An identifier that has been previously declared as a type in a ++DCL_TYPE++ operation.
timeint	Representation of a time interval
typeclass	Either BITS (meaning bitstring), REAL, or TIMEINT (meaning time interval).
version	A version name applicable to the given typeclass. Version names and characteristics are listed in Appendix 6.

EC.DATA.5 Dictionary:

Term	Definition
!+destination+!	variable, register or a list of such entities; will contain results of operation.
!+fall back value+!	programmer provided value to be used in case a division is not successful.
!+max div result+!	programmer's best estimate of the maximum result of a given division instruction.
!+radix pt ident+!	Interpreting the bitstring as a binary real number with bit 0 the most significant bit, 2 raised to the !+radix pt ident+! power is the significance of the rightmost (highest numbered) bit. For instance, a value of zero means that the bitstring represents an integer.
!+source+!	variable, register, literal or constant; has a value to be used as input to the operation.
!+user threshold+!	smallest difference user programs specify for a comparison operation; i.e., two numbers whose difference is less than this are considered equal.

EC.DATA.6 Undesired Event Dictionary:

%constant destination%	A user program has specified a constant or a literal as a destination.
%divide by zero%	A user program attempted to divide by zero.
%illegal array index%	The index supplied in an array reference is not in the index set of the array.
%illegal index set%	The index set of an array is not: (a) contiguous (b) in ascending order (c) integers (d) constants or literals.
%inappropriate attributes%	The attributes given are not valid for the type class at hand.

%inconsistent lengths%	An operation has been specified with bitstrings (or substrings) of different lengths.
%inconsistent register access%	An operation that changes the value of a register has the "-SAVE" suffix.
%left truncation%	The most-significant bits are lost in a real to bitstring conversion. This results from the user specifying a radix point too close to the most significant bit in the destination bitstring.
%list mismatch%	In a +SET+ operation, the number of given source operands differs from the number of given destination operands.
%name in use%	A name has been defined in two declarations.
%nonexistent position%	A user has specified (1) a start position that does not exist in the bitstring; or (2) a start position and a length that define a substring not contained in the bitstring.
%range exceeded%	The value being stored into a variable is outside the range of the variable.
%undeclared operand%	Operand has not been declared.
%undeclared spectype%	An entity has been declared using a specific type that has not been declared.
%unknown value%	A variable has been used in a declaration.
%wrong init value size%	The set of initial values is not the same size as the array.
%wrong init value type%	A constant or literal used as an initial value is not the type of the entity or array.

EC.DATA.7 <u>System Generation Parameters</u>: None

EC.DATA.8 Information Hidden

1. The representation of numeric objects in terms of hardware data types.
2. How range and resolution information are used to determine representation.
3. The procedures for performing numeric operations.
4. The representation of bitstrings.
5. How to access a bit within a bitstring.
6. The procedures used to perform bitstring operations.
7. How times are represented for the hardware timers.
8. How to compute the memory location of an array element given the array name and the element index.

EC.IO
INPUT/OUTPUT

EC.IO.1 Introduction

This module implements two types of bitstring entities known as input data items and output data items, which are used to communicate between the computer and external devices. This interface does not include facilities for i/o used during channel diagnostics (see EC.TEST.1).

Each data item may be enabled or disabled by user programs. When enabled, communication with the outside world is possible. The values of input data items may be set by external devices. The values of output data items are transmitted to external devices. When a data item is disabled, its connection with the outside world is severed.

Although input data items are normally "read-only" and output data items are normally "write-only", a few may be both read and written when they are disabled. These may be used as storage at such times. An input (output) data item may always be used as a source (destination) in an EC statement.

User programs are able to check to see if an external communication has been successful.

Some of the input is volatile. Programs will be notified when it is available and will have a limited amount of time to obtain the values.

Within these constraints, an input or output data item may be used exactly as other bitstring variables.

EC.IO.2 Interface Overview

EC.IO.2.1 Access programs

Program name	Parm type	Parm info	Undesired Events
+DISABLE+	p1:dataitem;I	name of data item	%not a data item% %already disabled%
+ENABLE+	p1:dataitem;I	name of data item	%not a data item% %already enabled%
+G_SUCCESS+	p1:dataitem;I p2:boolean;O	name of data item !+i/o success+!	%not a data item%

Program effects

+ENABLE+	Enables transmission to/from the external environment. If p1 is an input data item, it will now change value when an input transmission occurs. If p1 is an output data item, the value will now be transmitted externally each time it is changed. If the item is read-write input, use of the item as a destination in an EC statement is now prohibited until disabled. If the item is read-write output, use of the item as a source in an EC statement is now prohibited until disabled. At system-generation time, all data items are enabled.
+DISABLE+	Transmission to/from the external environment will be inhibited. If the item is read-write input, it may now be used as a destination in an EC statement. If the item is read-write output, it may now be used as a source in an EC statement.

EC.IO.2.2 Built-in Objects

The names of all data items are listed in an appendix to this document.

Undesired Events associated with Built-in Objects

The following undesired events may occur when data items are used in EC statements:

%read-write violation%

EC.IO.2.2 Events signalled by incrementing a semaphore

Some input data items may only be read intermittently. For each such data item, the EC signals an event (by incrementing a semaphore) when that item may be read. The event will be of the form

@T(!+ x ready+!)

where "x" is replaced by the name of the data item. The intermittent data items, and the semaphore that will be incremented when each becomes ready, are listed in an appendix to this document.

Event effects

An input transmission to the named input data item can only be expected to be successful within #data available x# (where "x" is replaced by the name of the data item) time after the event is signalled. At all other times, the !+i/o success+! boolean associated with that data item will be false.

EC.IO.3 Undesired Event Assumptions

1. User programs will not attempt to
   - use an enabled read-write input data item as a destination; or
   - use an enabled read-write output data item as a source.
2. User programs will not disable (enable) a data item that is already disabled (enabled).

EC.IO.4 Local Type Definitions

dataitem	The name of any input or output data item. The data item names are listed in an appendix to this document.

EC.IO.5 Dictionary

Term	Definition
Any item of the form !+ x ready+!	The named data item is now available for read operations.
!+i/o success+!	true iff the last transmission associated with the named data item was successful.

EC.IO.6 Undesired Event Dictionary

%already disabled%	A user program has tried to disable a data item already disabled.
%already enabled%	A user program has tried to enable a data item already enabled.
%not a data item%	A user program has specified a nonexistent data item.
%read-write violation%	A user program has illegally used a data item as a source or destination in violation of its current read-write capabilities. In particular, one of the following has occurred: - a read-only item was used as a destination; - a write-only item was used as a source; - an enabled read-write input item was used as a destination; - an enabled read-write output item was used as a source.

EC.IO.7 System-Generation Parameters

Parameter	Type	Explanation
#data available x#	timeint	(where "x" is replaced in turn by the name of each intermittent data item) Length of time that the named data item is guaranteed to remain available after @T(!+/ / ready+!) for that data item
#max i/o time x#	timeint	(where "x" is replaced by the name of each data item) The maximum time interval that can elapse between the beginning of the access program that reads/writes the named item, and the time it takes for the external transmission to take place.

EC.IO.8 <u>Information Hidden</u>

1. Hardware instruction sequences to perform I/O operations.
2. The technique used to prevent simultaneous use of resources.
3. How the module detects that intermittent data are ready to read.
4. The values written out for discrete output word bits that are unused.
5. The assignment of information to channels and I/O words.
6. How the success or failure of an I/O operation is determined.
7. When input operations actually take place.

EC.PAR.1
PROCESS MECHANISMS

EC.PAR.1.1 Introduction

The process mechanism allows the definition of a set of sequential processes that will proceed in parallel and unknown relative speeds. Demand processes are activated when specific events occur. Periodic processes may be turned on or off, but are re-started at regular intervals when turned on.

EC.PAR.1.2 Interface Overview

EC.PAR.1.2.1 Instruction Templates

Keywords	Parameters associated with keywords		Undesired events
++D_PROCESS++	p1:timeint;I	!+max CPU time req+!	%inconsistent time parms%
	p2:timeint;I	!+deadline+!	%max CPU time exceeded%
	p3:program name;I	process body	
++P_PROCESS++	p1:timeint;I	!+max CPU time req+!	%inconsistent time parms%
	p2:timeint;I	!+deadline+!	%illegal synch%
	p3:timeint ent;I	!+period+!	%undeclared semaphore%
	p4:semaphore;I	!+starting event+!	%max CPU time exceeded%
ON_OFF	p5:boolean ent;I_OPT	!+on/off+!	%missed deadline%
	p6:program name;I	process body	

Parameters

p1 and p2 must be given by a literal or constant.

Instruction effects

++D_PROCESS++ establishes a demand process that becomes active after @T(!+power up+!). The body of the process is the program named by p3. The process remains active until it is suspended as a result of a synchronization operation or executes the last statement in its body. During the interval when it is active, it will be given up to p1 CPU time before p2 real time has elapsed. A process that is suspended as a result of a synchronization operation may start again. A process that executes its last statement will start again only after a system generation.

++P_PROCESS++	establishes a periodic process that becomes active after the semaphore named by !+starting event+! becomes nonnegative. The body of the process is the program named by p6. While the on/off boolean named by p5 is true (or always if no boolean is supplied), a built in semaphore, NEXT_PERIOD, will be incremented at the start of each !+period+! amount of real time. After the start of a !+period+!, the process will be given up to p1 CPU time before p2 real time has elapsed. The process must complete execution and perform +DOWN+(NEXT_PERIOD) +PASS+(NEXT_PERIOD) before consuming more than p1 CPU time. If boolean entity given as p5 becomes false during execution, the process will stop when it waits for the start of its next !+period+! (by doing +PASS+(NEXT_PERIOD)).
Both	If two (or more) processes simultaneously execute sequences of statements that read and/or alter the value of some data, the results are unpredictable because the executions may overlap in time. However, EC access programs are considered indivisible. If two EC access programs are executed simultaneously by two processes, the effect will be as if one of the processes executed its access program before the other; the order is not specified. Note that the invocation of a user-supplied routine is the execution of a single EC access program, but the execution of the body of that routine is a sequence of EC statements.

EC.PAR.1.2.2 Built-in objects

Name	Used to Refer to
NEXT_PERIOD	semaphore variable, private to each periodic process, that will be incremented by the EC at the start of each period. Each periodic process only has access to its own NEXT_PERIOD. Semaphores are described in EC.PAR.3.

EC.PAR.1.3 Undesired Event Assumptions

1. Each program will complete its processing in the time specified.

2. Demand processes will not need to run so often as to cause a periodic process to miss its deadline.

EC.PAR.1.4 Local Type Definitions

boolean ent	The name of a previously-declared boolean variable or constant.
program name	See EC.SEQ.2.4.
timeint ent	The name of a previously-declared timeint variable or constant.

EC.PAR.1.5 Dictionary

Term	Definition
!+deadline+!	The maximum amount of real-time that can be allowed to elapse between the time that a process can proceed and the time that it reaches the next point of suspension.
!+max CPU time req+!	An upper bound for the amount of CPU time required by a process before it will suspend itself. The value of this parameter is machine dependent.
!+on/off+!	the name of a boolean variable or constant whose value will be used to start/stop the periodic process in whose definition it appears. Its value must be true whenever the periodic process is supposed to proceed. If it becomes false, the process will be suspended the next time it reaches its starting point.
!+period+!	The name of a timeint entity whose value will be interpreted as the amount of real-time that should elapse between the beginning of one execution of a periodic process and the beginning of the next execution. If the named entity is a variable, changing its value has the result of changing the period of any process for which it was used as the !+period+!.
!+starting event+!	The name of a semaphore that, when becoming nonnegative, will cause the periodic process in which it is named to become active.

EC.PAR.1.6 Undesired Event Dictionary

%illegal synch%	a synchronization operation other than the required +DOWN+(NEXT_PERIOD) and +PASS+(NEXT_PERIOD) (see EC.PAR.3) appears in the body of a periodic process; or those required operations were omitted.
%inconsistent time parms%	the timing parameters are contradictory; e.g. !+max CPU time req+! exceeds !+deadline+!, or !+deadline+! exceeds the current value of !+period+!.
%max CPU time exceeded%	a process has used more CPU time than the maximum that was specified in its declaration.
%missed deadline%	a process has missed its deadline because too many demand processes have occurred.
%undeclared semaphore%	User program has used a semaphore that has not been declared.

EC.PAR.1.7 System Generation Parameters: None

EC.PAR.1.8 Information Hidden

1. How many processors there are, and how the processes are allocated among them.
2. The data structures and operations required to load a process on a processor.
3. The data structures required to represent processes and keep track of their current states.

EC.PAR.2
EXCLUSION REGIONS

EC.PAR.2.1 Introduction

This module allows constraints to be placed on the potential concurrency of processes executing regions of code by defining an exclusion relation among them. Region 1 excludes region 2 if starting to execute region 2 is forbidden while region 1 is being executed. Mutual exclusion is a special case of this exclusion relation, which is based on [BELP73].

EC.PAR.2.2 Interface Overview

EC.PAR.2.2.1 Instruction Template and Access Programs

Keywords	Parameters associated with keywords		Undesired events
++REGION++	p1:name;I	region name	%name in use%
	p2:statement-list;I	region body	
END-REGION			

Program name	Parm type	Parm info	Undesired events
++EXCLUSION++	p1:exclusion-relation;I		%undeclared region%

Instruction and program effects:

++EXCLUSION++ If the exclusion relation includes (A,B) then no process will begin to execute region B in the interval that starts when a process begins execution of region A and ends when that process completes execution of region A.

++REGION++ p1 may be used to stand for the section of code that is given in p2. If the last action before the region causes a process to wait, then the process is considered to be inside the region when it is allowed to proceed. If the last instruction in p2 is a wait operation, then the process is considered to have left the region when it begins to wait. Including regions in a process will prevent the process from waking up, if doing so would result in a violation of an exclusion region.

EC.PAR.2.3 Undesired Event Assumptions: None.

EC.PAR.2.4 Local Type Definitions:

exclusion-relation	a parenthesized list of ordered pairs of region names; the ordered pairs are separated by commas. The first region named in each pair excludes the second region named in that pair.

EC.PAR.2.5 Dictionary: None

EC.PAR.2.6 Undesired Event Dictionary

%undeclared region%	an exclusion relation includes regions that have not been identified in the program.
%name in use%	a ++REGION++ instruction has been given using a previously defined identifier

EC.PAR.2.7 System Generation Parameters: None

EC.PAR.2.8 Information Hidden

1. How the exclusion relation is implemented.

EC.PAR.3
SYNCHRONIZATION VARIABLES AND OPERATIONS

EC.PAR.3.1 Introduction

This module provides a run-time synchronization mechanism, semaphores, with associated operations. They can be used where exclusion regions cannot express the constraints. This mechanism is based on [BELP73]; the semaphore operations are a more primitive version of Dijkstra's P and V [DIJK68].

EC.PAR.3.2 Interface Overview

EC.PAR.3.2.1 Creating a Semaphore

To create specific semaphore types, use the ++DCL_TYPE++ program specified in EC.DATA.2, with:

p1 as described there;
p2 = SEMAPHORE;
p3 a semaphore-attribute, defined in EC.PAR.3.4;
p4 as described there; and
p5 omitted.

Semaphore entities must be declared before they can be used. Use the ++DCL_ENTITY++ program of Section EC.DATA.2.2, with:

p1 as described there;
p2 as described there;
p3 (if supplied) a semaphore-attribute;
p4 as described there; and
p5 the initial value given as an integer literal

EC.PAR.3.2.2 Access programs

Program name	Parm type	Parm Info	Undesired Events
+DOWN+ +UP+	p1:semaphore;IO		%undeclared semaphore% %constant destination% %range exceeded%
+PASS+	p1:semaphore;I		%undeclared% semaphore%

Program effects

In this section, we characterize informally the effects of the synchronization operations. For a more precise description, see the formal specifications in [TRACE].

Terminology:

Term	Explanation
state	Either "active" or "suspended".
state(self)	the state of the process executing the operation
state(waiters)	the state of all processes in the middle of a +PASS+ operation for that semaphore

Operation	Effect on the integer equivalent of the named semaphore	Effect on process state(s)
+UP+	incremented by 1	if the semaphore gteq 0 then state(waiters) := active*
+PASS+	none	if the semaphore lt 0 then state(self) := waiting
+DOWN+	decremented by 1	none

* a change in "state(waiters)" means that all the other processes in pending +PASS+ operations on that semaphore may be made active in an unspecified order. A process that becomes active may make the semaphore negative, causing any other processes in the midst of a +PASS+ to remain in the waiting state. Processes will be activated and complete the pass as long as the semaphore is nonnegative.

EC.PAR.3.3 Undesired Event Assumptions

1. There is a range of values that will suffice for all semaphores, and will not be exceeded by user programs.

EC.PAR.3.4 Local Type Definitions

semaphore	The name of a run-time synchronization object created previously by a user program; or NEXT PERIOD, the builtin semaphore private to each periodic process, defined in EC.PAR.1.2.2.
semaphore-attribute	An ordered pair of integers specifying the lower bound and upper bound of the type.

EC.PAR.3.5 Local dictionary: None

EC.PAR.3.6 Undesired Event Dictionary:

%constant destination%	A user program has tried to change the value of a semaphore declared to be a constants
%range exceeded%	A user program has caused a semaphore variable to exceed its maximum or minimum value range.
%undeclared semaphore%	A user program contains a synchronization operation on an identifier that was not declared as a semaphore.

EC.PAR.3.7 System Generation Parameters: None.

EC.PAR.3.8 Information Hidden

1. The data structures and algorithms required to keep track of semaphore values and of process states.

2. How synchronization operations are made indivisible.

EC.SEQ.1
STATEMENT SEQUENCE CONTROL CONSTRUCTS

EC.SEQ.1.1 Introduction

The statement sequence control constructs allow the programmer to control the order of actions within a program or process. It is based on the deterministic version of the it ti control structure, which has been formally designed in [ITTI2]. The following is an informal introduction, which should be read in conjunction with the syntactic definitions on the next page. We assume the availability of sequences of statements (defined in EC.SEQ.1.4) and a special class of such statements, called boolean sequences, which evaluate predicates without changing the state of variables.

An LP defines a limited program; it limits the states in which a statement list will be executed to those in which the guard (see below) is true.

An LPL (limited program list) is an optional guard definition (see below) followed by a list of limited programs; when the LPL is executed, the first limited program in the list that has a true guard will be executed. The behavior of an LPL when there is no true guard is undefined.

Guards may be (1) boolean sequences, (2) limited program lists, or (3) guard definition programs.

A boolean sequence is a statement list that changes the value of the built-in boolean variable GUARDVAL. The value of a boolean sequence is the value that it assigns to GUARDVAL. The semantics do not define the meaning of boolean sequences that do not assign a value to GUARDVAL. They do not fully define the meaning of boolean sequences that changes values other than GUARDVAL.

A guard that is an LPL is considered true if and only if the guard of one of its elements is true.

A guard definition program (DEF) defines a list of identifiers that can be used as guards. The body of a DEF is a DLPL, which is a list of DLPs (definition limited programs). A DLP is exactly like an LP except that, in place of the statement list, there is either an identifier, the meaning of which is being defined, or a DLPL. By executing the body of a DEF, one of the identifiers will be selected. The identifiers, called defined guards, are considered true in the states where they will be selected.

The IT TI brackets enclose a repeatable statement list. A decision about whether or not to repeat the execution of that list may be made at any LP or DLP within the list except those within an enclosed IT TI. The decision is indicated by means of a key that indicates whether to continue the iteration (CONT) or stop (TERM). The key in the last selected LP or DLP in the IT TI determines whether or not iteration will take place. Any key in an LP or DLP selected earlier has no effect.

The SKIP instruction has no effect on the state of the program. It is to be used instead of an empty statement list.

EC.SEQ.1.2 Interface Overview

EC.SEQ.1.2.1 Instruction Templates

In the following table, the appearance of a keyword in the parameter column means that the parameter is an instruction beginning with that keyword.

Keywords	Parameters associated with keywords	Undesired Events
+LP+		
	p1:guard;I	%illegal guard%
	p2:statement-list;I	%key outside IT TI%
	p3:key;I_OPT	
PL		
+LPL+		
	p1:+DEF+;I_OPT	%guard unused%
	p2-pn:+LP+;I	%no appropriate action%
LPL		
+DEF+		
	p1:identifier-list;I	%previously defined id%
	p2:+DLPL+;I	%undefined guard%
FED		%guard unused%
+DLP+		
	p1:guard;I	%illegal guard%
	p2:defitem;I	%key outside IT TI%
	p3:key;I_OPT	%not guard id%
PLD		
+DLPL+		
	p1:+DEF+;I_OPT	%guard unused%
	p2-pn:+DLP+;I	%no appropriate action%
LPLD		
+IT+		
	p1:statement-list;I	%key missing%
TI		
+SKIP+	—	None

Instruction Effects

The instructions are explained informally in the introduction. A formal definition may be found in [ITTI2].

EC.SEQ.1.2.3 Built-in Objects

Name	Refers to	Undesired Event
GUARDVAL	a special boolean entity, which stores the value of a boolean-seq (see EC.SEQ.1.4).	%illegal GUARDVAL use%
INIT	a boolean entity that is true on the first execution of the body of an IT TI after the IT TI is entered, but false on subsequent iterations of the body.	%illegal INIT use%

EC.SEQ.1.3 Undesired Event Assumptions

1. Except when an LPL is used as a guard, every LPL instruction will include an executable statement list for every state in which the LPL would be executed.

2. User programs will not attempt to use GUARDVAL as an operand outside the context of an LP or DLP. When GUARDVAL is used, user programs will not neglect to give GUARDVAL a value. They will treat the initial value of GUARDVAL as undefined.

3. There is no way to execute the body of an IT TI without executing an element with a key.

EC.SEQ.1.4 Local Type Definitions

boolean-seq	a sequence of statements resulting in a boolean value being stored in GUARDVAL. The sequence of statements is not a boolean-seq if it fails to set the value of GUARDVAL. If a boolean-seq that evaluates to false changes the state of a variable, the value of that variable is not defined outside the guard in which the boolean-seq appears.
defined guard	identifier that has appeared in DEF for this LPL or DLPL
defitem	either an identifier or a DLPL
guard	boolean-seq or LPL or defined guard
identifier-list	one or more identifiers separated by commas; the list is enclosed in parentheses.
key	either TERM or CONT
statement-list	a sequence of EC instructions, invocations of EC access programs, and invocations of user-defined EC programs.

EC.SEQ.1.5 Dictionary: None

EC.SEQ.1.6 Undesired Event Dictionary

%guard unused%	A guard has been defined in a DEF but is not used in the LPL or DLPL that encloses the definition
%illegal guard%	Either (1) a boolean-seq does not assign a value to GUARDVAL or (2) the guard is not a boolean-seq, an LPL, or an identifier that has been defined as a guard by a DLPL
%illegal GUARDVAL use%	GUARDVAL is not used in a guard of an LP or DLP, or is not assigned a value in a boolean-seq where it appears, or its initial value is not treated as undefined.
%illegal INIT use%	INIT used outside of guards in an IT TI

%key missing%	an IT TI has been written such that it is possible to execute its body without selecting a key.
%key outside IT TI%	key specified outside of an IT TI
%no appropriate action%	none of the guards in an LPL that must be executed are true.
%not guard id%	identifier in a DLP is not included in the guard set of the enclosing DEF
%undefined guard%	one or more guards in the identifier list have not appeared in a DLP

EC.SEQ.1.7 System Generation Parameters: None

EC.SEQ.1.8 Information Hidden

1. The control structures that exist at the hardware level.

2. The hardware instruction sequences needed to implement the EC control structures.

EC.SEQ.2
PROGRAM INVOCATION FACILITIES

EC.SEQ.2.1 Introduction

The Program Invocation module provides mechanisms for declaring and invoking programs with parameters.

EC.SEQ.2.2 Interface Overview

EC.SEQ.2.2.1 Instruction Templates and Access Programs

Keywords	Parameters associated with keywords		Undesired Events
++PROGRAM++	p1:name;I	program name	%parm access violation%
	p2:call-sort;I	how invoked	
PARM	p1:name;I	formal parameter name	
	p2:type-spec;I	parameter type required	
	p3:access-spec;I	is parm input or output, optional or required	
. . .	. . .	. . .	
PARM	p1:name;I	formal parameter name	
	p2:type-spec;I	parameter type required	
	p3:access-spec;I	is parm input or output, optional or required	
UE	p1:name;I	name of undesired event	%duplicate UEs%
. . .	. . .	. . .	
UE	p1:name;I	name of undesired event	
BEGIN			
	p1:statement-list;I	program body	
END			

Parameters

If the program has no formal parameters, then the PARM keywords and parameters are omitted. If the program has no undesired events associated with it, then the UE keywords and parameters are omitted.

Program	Parm type	Parm info	Undesired events
+program-name+	(p1, . . ., pn)	parameter list required by program	%too few parms% %parm wrong type% %too many parms%
	(program name, . . ., program name)	list of UE-handling programs (optional)	%undeclared program% %UE-handler has parameters% %no UE correspondence%
++RANK_PGM++	p1:program name;I p2:program name;I	program p1 should be invoked no slower than program p2	%undeclared program%
%UE name%	--	invoke program to handle the named UE	%undeclared UE%

Instruction and program effects

++PROGRAM++ The program p1 may be invoked by any program that uses its name. The invocation will be by inline expansion of the body if p2 = MAC, and by subroutine invocation and return if p2 = SUB. Supplying a list of UE names requires that when the program is invoked, a program must be named to handle each UE.

+program-name+ (where "program name" is replaced with the name of a program previously defined by using the ++PROGRAM++ facility) The effect is as if the statement list that is the body of the program (as declared using ++PROGRAM++) is (1) modified by replacing the formal parameters with the description of the entity, array, or array element that is the actual parameter and then (2) inserted in place of +program-name+. If the resulting program modifies the value of a variable that was used to determine which element of an array is the actual parameter, the effect is undefined. If an optional parameter is omitted, any commas that would precede it and follow it (were it supplied) must be given, unless no other parameters trail it in the list; then the trailing comma may be omitted.

The program names will be associated with the UE's specified when the program was declared; the correspondence is positional. If the UE-program list is provided, there must be one program name for every UE name. If the list is omitted, the correspondence between UE names and UE programs is that defined by the textually most recent call of the +program+. Note that the UE-program list must be supplied at the earliest textual point where the routine is called. A UE-handling program is not allowed to have formal parameters.

++RANK_PGM++	Expresses the preference that the time it takes to invoke the program named by p1 will not be more than the time it takes to invoke the program named by p2. The Extended Computer will order the programs' invocation speeds based on these preferences.
%UE name%	(where "UE name" is replaced with the name of an undesired event program that was declared when the enclosing program was defined using the ++PROGRAM++ facility). Causes the UE-handling program associated with this UE to be invoked. The association is determined by the caller of the enclosing program at invocation time.

EC.SEQ.2.2.2 Built-in Objects

Name	Refers to	Undesired Events
PARM_GIVEN	An array of booleans private to each program. There is one element for each formal parameter that was specified when the program was declared. Correspondence is positional; the first parameter corresponds to the first element of the array, etc. When the program is invoked, an element of PARM_GIVEN will be true if the corresponding actual parameter was given in the invocation, and false if it was omitted. PARM_GIVEN may not be referenced outside the body of a ++PROGRAM++ instruction.	%illegal PARM_GIVEN use%

EC.SEQ.2.3 Undesired Event Assumptions

1. When user programs invoke a program, they will not supply parameters whose types are not consistent with the specification in the template;

2. A program will not neglect to assign a value to an output parameter.

EC.SEQ.2.4 Local Type Definitions

access-spec

	Input	Output	Input and Output
Required:	I	O	IO
Optional:	I_OPT	O_OPT	IO_OPT

I:	input parameters, treated as constants;
O:	output parameters, treated as uninitialized variables;
IO:	input-output parameters, treated as previously initialized variables.

call-sort	Either SUB meaning subroutine linkage or MAC meaning inline expansion, i.e. macro.
program name	The name of a program declared via the ++PROGRAM++ statement.
type-spec	Either: 1) .. the name of any type class or specific type. (see EC.DATA) 2) .. the preceding, followed by "ARRAY" (See EC.DATA).

EC.SEQ.2.5 Dictionary: None.

EC.SEQ.2.6 Undesired Event Dictionary

%duplicate UEs%	A UE has been named more than once in a program declaration.
%illegal PARM_GIVEN use%	PARM_GIVEN is referenced outside the body of a ++PROGRAM++ instruction, or in the body of a program with no optional parameters, or with an index that is less than one or greater than the number of optional parameters.
%no UE correspondence%	The first (textually) invocation of a program has been issued without specifying the UE-handlers for that program.
%parm access violation%	Use of a parameter inside a program does not match the access specification given for it in the program declaration; i.e., the value of an input parameter was changed, or the value of an output parameter was not assigned.

%parm wrong type%	The type of a parameter, as supplied by a calling program, is inconsistent with the specification in the called program's parameter list.
%too few parms%	User program fails to provide one or more of the parameters required by the called program.
%too many parms%	The number of actual parameters supplied by the user is greater than that specified by the called program.
%UE-handler has parameters%	For a UE-handler, a program with formal parameters has been specified; this is not allowed.
%undeclared program%	Program called or referenced has never been declared with ++PROGRAM++.
%undeclared UE%	Program has tried to invoke a UE-handler program by referencing a UE that was never named when the program was declared.

EC.SEQ.2.7 System Generation Parameters: None

EC.SEQ.2.8 Information Hidden

1. Whether a program is implemented as a subroutine or a macro is hidden from the caller of the program.

2. How control gets transferred to the program and later returned to the user program including any subroutine linkage conventions, such as saving and restoring registers.

3. How parameter information is communicated between the user program and the program.

EC.SEQ.3
TIMER FACILITIES

EC.SEQ.3.1 Introduction

This module provides facilities for measuring real time intervals via timers. A timer is a timeint variable that, when running, will increment or decrement at a rate commensurate with real time.

A timer may be used anywhere a timeint variable may be used, but there are two additional operations, START_TIMER and HALT_TIMER, that may be used. START_TIMER increments or decrements the timer until a limit is reached.

When a timer is declared, the user may choose between timers that increment and timers that decrement, as well as between timers that halt when they reach their limit and timers that "wrap around". The user may also specify a semaphore that will be incremented when the timer reaches its limit.

EC.SEQ.3.2 Interface Overview

EC.SEQ.3.2.1 Declaring a Timer

Timers are a numeric type class, as described in EC.DATA.1. To declare specific timer types, use the ++DCL_TYPE++ program specified in EC.DATA.2, with:

p1 as described there;
p2 = TIMER;
p3 a timer-attribute, defined in section EC.SEQ.3.4;
p4 as described there; and
p5 omitted.

No specific type may have a resolution less than #min timer resolution#, or else the undesired event %res too fine% will be raised.

Timer entities must be declared before they can be used. Use the ++DCL_ENTITY++ program of Section EC.DATA.2.2, with:

p1 as described there;
p2 as described there;
p3 (if supplied), a timer-attribute;
p4 = VAR; and
p5 the initial value given as a timeint literal

EC.SEQ.3.2.2 Access programs

Program name	Parm type	Parm Info	Undesired Events
++TIMER_EVENTS++	p1:timer;I	timer name	%undeclared timer%
	p2:semaphore;I	limit value event	%undeclared semaphore%
+START_TIMER+	p1:timer;I	timer name	%undeclared timer%
+HALT_TIMER+			

Program Effects

++TIMER_EVENTS++	Causes an event to be signalled (by incrementing p2) every time p1 reaches its minimum range value (if p1 is a decrementing timer) or its maximum range value (if p1 is an incrementing timer).
+START_TIMER+	Causes the value of p1 to be changed in value in real time. The value will be increased or decreased according to the declaration of the specific type to which p1 belongs. According to the declaration of the specific timer type to which p1 belongs, the timer will either stop when it reaches its minimum (maximum) value, or "wraparound"; i.e., continue from its maximum (minimum) value. Starting a running timer has no effect.
+HALT_TIMER+	Causes running timer p1 to halt. Halting a non-running timer has no effect.

EC.SEQ.3.3.2 Undesired Event Assumptions: None.

EC.SEQ.3.4 Local Type Definitions

timer	The name of a time-keeping mechanism declared previously by a user program.
timer-attribute	An ordered 5-tuple of the form (timeint, timeint, timeint, STOP/WRAP, UP/DOWN) The first three elements specify the lower bound, upper bound, and minimum resolution, respectively, of entities of the type. The fourth element is either "STOP" (meaning that the timer should stop when it reaches a limit) or "WRAP" (meaning that the timer should wrap around when it reaches a limit). The fifth element is either "UP" (meaning that the timer increments when started) or "DOWN" (meaning that the timer decrements when started).

EC.SEQ.3.5 Dictionary: None.

EC.SEQ.3.6 Undesired Event Dictionary

%res too fine%	User program sought to declare a specific timer type with a resolution finer that the minimum allowed.
%undeclared semaphore%	User program has used a semaphore that has not been declared.
%undeclared timer%	User program has used a timer that has not been declared.

EC.SEQ.3.7 System Generation Parameters

Parameter	Type	Explanation
#max timer error#	real	maximum allowable error rate of all timers, given as a fraction of the time interval measured
#min timer resolution#	timeint	for all timers, the minimum resolution

EC.SEQ.3.8. Information Hidden

1. The number of actual hardware timing devices and their characteristics, including range and resolution.

2. Data structures needed to keep track of all the active timers, and algorithms needed to process all of them.

3. Which hardware timing device is assigned to a timing task.

EC.STATE
EXTENDED COMPUTER STATE

EC.STATE.1 Introduction

This module controls and reports transitions between Extended Computer states.

EC.STATE.2 Interface Overview

EC.STATE.2.1 Access programs

Program name	Parm type	Parm info	Undesired Events
+S_FAIL_STATE+	—		None

EC.STATE.2.2 Events signalled by incrementing a semaphore

Event	Semaphore incremented when event occurs
@T(!+power up+!)	ECPOWUP
@T(!+failed state+!)_	ECFAILED

Program and Event effects

+S_FAIL_STATE+	The Extended computer enters its failed state, increments ECFAILED, and executes an internal shutdown procedure.
@T(!+failed state+!)	Programmers should assume that when #close down time# has elapsed after this event, no more software actions can occur.
@T(!+power up+!)_	The Extended Computer has entered the operating state and is functioning correctly. All demand processes are started.

EC.STATE.3 Undesired Event Assumptions: None.

EC.STATE.4 Local Type Definitions: None.

EC.STATE.5 Dictionary

Term	Definition
!+power up+!	computer is in the operating state and may be assumed to be functioning properly.
!+failed state+!	computer is malfunctioning.

EC.STATE.6 Undesired Event Dictionary: None.

EC.STATE.7 System Generation Parameters:

Parameter	Type	Explanation
#close down time#	timeint	The minimum expected time interval between the moment that the extended computer enters failed state and the moment when no more software actions may occur.

EC.STATE.8 Information Hidden

1. How the hardware behaves when the power is turned on.
2. How the hardware behaves when it enters malfunction states.
3. How malfunctions in hardware functions are detected and reported.
4. How many actual states the hardware has, and what hardware transitions are possible.
5. What internal malfunctions cause transitions from operating to failed.
6. What causes transitions to off, and to operating.

EC.TEST
EXTENDED COMPUTER DIAGNOSTIC TESTS

EC.TEST.1 Introduction

This module provides diagnostic commands that can be used to check the reliability of the memory, input/output hardware, and the timer and interrupt hardware. Invoking these programs may interfere with other programs as described below.

EC.TEST.2. Interface Overview

EC.TEST.2.1 Access Programs

EC.TEST.2.1.1 Input/Output Diagnostics

Program name	Parm. type	Parm. info	Undesired Events
+TEST_AC+	pl:boolean;O	!+test result+!	None
+TEST_CSA+	pl:boolean;O	!+test result+!	
+TEST_CSB+	pl:boolean;O	!+test result+!	
+TEST_DC+	pl:boolean;O	!+test result+!	
+TEST_DIOW1+	pl:boolean;O	!+test result+!	
+TEST_DIOW2+	pl:boolean;O	!+test result+!	
+TEST_DIOW3+	pl:boolean;O	!+test result+!	
+TEST_XACC+	pl:boolean;O	!+test result+!	
+TEST_YACC+	pl:boolean;O	!+test result+!	
+TEST_ZACC+	pl:boolean;O	!+test result+!	

EC.TEST.2.1.2 Memory Diagnostics

+TEST_MEMORY+	pl:boolean;O	!+test result+!	None

EC.TEST.2.1.3 Timer/Interrupt Diagnostics

+TEST_TIMER+	pl:boolean;O	!+test result+!	None
+TEST_INTERRUPTS+	pl:boolean;O	!+test result+!	

EC.TEST.2.3 Program and Event effects

These programs set !+test result+! = true if and only if the hardware passed the associated test. If the test is performed periodically or independent of user request, the result given will be that of the most recent test. If the test is performed on request, the command will initiate the test and report the result when the test is complete. In addition, the following effects are observable.

EC.TEST.2.3.1 Input/Output Diagnostics

+TEST_AC+ This program reports the results of the AC signal converter check. It may interfere with:

output, when the data item is

```
//BRGDEST//   //GNDTRK//
//RNGHND//    //RNGTEN//     //RNGUNIT//
//STEERAZ//   //STEEREL//
```

+TEST_CSA+ This program reports the results of the cycle-steal channel A and serial channel 1 check. It may interfere with:

output, when the data item is

```
//ASAZ//       //HUDCTL//     //USOLCUAZ//
//ASEL//       //LSOLCUAZ//   //USOLCUEL//
//ASLAZ//      //LSOLCUEL//   //VERTVEL//
//ASLEL//      //MAGHDGH//    //VTVELAC//
//ASLCOS//     //MAPOR//      //XCOMMF//
//ASLSIN//     //PTCHANG//    //XCOMMC//
//AZRING//     //PUACAZ//     //YCOMM//
//BAROHUD//    //PUACEL//
//FLTDIRAZ//   //ROLLCOSH//
//FPMAZ//      //ROLLSINH//
//FPMEL//
```

input, when the data item is /LOCKEDON/ or /SLTRNG/.

+TEST_CSB+ This program reports the results of the cycle steal channel B and serial channel 2 check. It may interfere with:

output, when the data item is

```
//CURAZCOS//  //CURAZSIN//   //CURPOS//
```

input, when the data item is

```
/ANTGOOD/     /DGNDSP/       /DRFTANG/
/DRSFUN/      /DRSMEM/       /DRSREL/
/ELECGOOD/
```

+TEST_DC+

This program reports the results of the DC signal converter check. It may interfere with:

output, when the data item is

```
//FPANGL//     //GNDTRVEL//     //STERROR//
```

+TEST_DIOW1+
+TEST_DIOW2+
+TEST_DIOW3+

These programs report the results of the checks on discrete input and output word pairs. 1, 2, and 3 respectively. These programs may interfere with:

output, when the data item is

```
//DOW1//       //DOW2//
```

input, when the data item is

```
/DIW1/         /DIW2/          /DIW3/
/DIW4/         /DIW5/          /DIW6/
/ANTGOOD/      /DGNDSP/        /DRFTANG/
/DRSFUN/       /DRSMEM/        /DRSREL/
/ELECGOOD/     /LOCKEDON/      /SLTRNG/
/SINSDD/
```

+TEST_XACC+
+TEST_YACC+
+TEST_ZACC+

These programs report the results of checks on the accelerometer and torque registers associated with the X, Y, and Z axes of the IMS respectively. These programs may cause the IMS to lose its alignment and velocities, and may interfere with:

output, when the data item is

```
//XGYCOM//     //YGYCOM//      //ZGYCOM//
```

input, when the data item is

```
/XGYCNT/       /XVEL/          /YGYCNT/
/YVEL/         /ZGYCNT/        /ZVEL/
```

EC.TEST.2.3.2 Memory Diagnostics

+TEST_MEMORY+

This program reports the results of the the memory diagnostic program. It interferes with no other program.

EC.TEST.2.3.3 Timer/Interrupt Diagnostics

+TEST_TIMER+
+TEST_INTERRUPTS+

These programs report the results of the timer and interrupt hardware checks. They may interfere with normal operation of timers and input/output commands in unpredictable ways.

EC.TEST.3 Undesired event assumptions: None.

EC.TEST.4 Local Type Definitions: None.

EC.TEST.5 Dictionary

Term	Definition
!+test result+!	true iff hardware passes built-in test.

EC.TEST.6 Undesired Event Dictionary: None.

EC.TEST.7 System Generation Parameters: None.

EC.TEST.8 Information Hidden

1. How the tests are performed and criteria used to judge results.
2. Timing characteristics that affect test evaluation.
3. Which parts of memory are checked.
4. The algorithm used to check memory.

EC.INDEX INDICES TO THE DOCUMENT

This section provides the following indices to the facilities described in this document:

Access programs

Instructions and keywords

Builtin objects

Events signalled by incrementing a semaphore

Types provided

Dictionary terms

Undesired events

System generation parameters

Access Programs

Access program	Where defined
+ABSV+	EC.DATA
+ADD+	EC.DATA
+AND+	EC.DATA
+B_REAL_2COMP+	EC.DATA
+B_REAL_POSITIVE+	EC.DATA
+B_REAL_SIGNMAG+	EC.DATA
+CAT+	EC.DATA
+COMPLE+	EC.DATA
++DCL_ARRAY++	EC.DATA
++DCL_ENTITY++	EC.DATA
++DCL_TYPE++	EC.DATA
+DISABLE+	EC.IO
+DIV+	EC.DATA
+DOWN+	EC.PAR.3
+EQ+ (bitstring)	EC.DATA
+EQ+ (numeric)	EC.DATA
+ENABLE+	EC.IO
++EXCLUSION++	EC.PAR.2
+G_SUCCESS+	EC.IO
+GEQ+	EC.DATA
+GT+	EC.DATA
+HALT_TIMER+	EC.SEQ.3
+LEQ+	EC.DATA
+LT+	EC.DATA
+MINUS+	EC.DATA
+MUL+	EC.DATA
+NAND+	EC.DATA
+NEQ+ (bitstring)	EC.DATA
+NEQ+ (numeric)	EC.DATA
+NOT+	EC.DATA
+OR+	EC.DATA
+PASS+	EC.PAR.3
+R_BITS_2COMP+	EC.DATA
+R_BITS_POSTIIVE+	EC.DATA
+R_BITS_SIGNMAG+	EC.DATA
+R_TIME_HOUR+	EC.DATA
+R_TIME_MIN+	EC.DATA
+R_TIME_MS+	EC.DATA
+R_TIME_SEC+	EC.DATA
++RANK_DATA++	EC.DATA
++RANK_PGM++	EC.SEQ.2
+REPLC+	EC.DATA
+S_FAIL_STATE+	EC.STATE

Access Programs (continued)

Access program	Where defined
+SET+	EC.DATA
+SHIFT+	EC.DATA
+START_TIMER+	EC.SEQ.3
+SUB+	EC.DATA
+T_REAL_HOUR+	EC.DATA
+T_REAL_MIN+	EC.DATA
+T_REAL_MS+	EC.DATA
+T_REAL_SEC+	EC.DATA
+TEST_AC+	EC.TEST
+TEST_CSA+	EC.TEST
+TEST_CSB+	EC.TEST
+TEST_DC+	EC.TEST
+TEST_DIOW1+	EC.TEST
+TEST_DIOW2+	EC.TEST
+TEST_DIOW3+	EC.TEST
+TEST_INTERRUPTS+	EC.TEST
+TEST_MEMORY+	EC.TEST
+TEST_TIMER+	EC.TEST
+TEST_XACC+	EC.TEST
+TEST_YACC+	EC.TEST
+TEST_ZACC+	EC.TEST
++TIMER_EVENTS++	EC.SEQ.3
+UP+	EC.PAR.3
+XOR+	EC.DATA
Invoking a user-defined access program	EC.SEQ.2

Instructions and Keywords

The terms bracketed with "+" or "++" are instructions; other terms are keywords associated with instructions.

Name	Where defined
BEGIN	EC.SEQ.2
+DEF+	EC.SEQ.1
+DLP+	EC.SEQ.1
+DLPL+	EC.SEQ.1
++D_PROCESS++	EC.PAR.1
END	EC.SEQ.2
END-REGION	EC.PAR.2
FED	EC.SEQ.1
+IT+	EC.SEQ.1
+LP+	EC.SEQ.1
+LPL+	EC.SEQ.1
LPL	EC.SEQ.1
LPLD	EC.SEQ.1
ON_OFF	EC.PAR.1
PARM	EC.SEQ.2
PL	EC.SEQ.1
PLD	EC.SEQ.1
++PROGRAM++	EC.SEQ.2
++P_PROCESS++	EC.PAR.1
++REGION++	EC.PAR.2
-SAVE	EC.DATA
+SKIP+	EC.SEQ.1
TI	EC.SEQ.1
UE	EC.SEQ.2

Builtin Objects

Name	Where defined
GUARDVAL	EC.SEQ.1
INIT	EC.SEQ.1
NEXT_PERIOD	EC.PAR.1
PARM_GIVEN	EC.SEQ.2
REG	EC.DATA
All input and output data items	EC.IO

Events Signalled by Incrementing a Semaphore

Event	Semaphore	Where defined
@T(!+/ENTERSW/ ready+!)_-	ENTSWSEM	EC.IO
@T(!+failed state+!)	ECFAILED	EC.STATE
@T(!+/KBDENBL/ ready+!)_-	ENBLSEM	EC.IO
@T(!+/KBDINT/ ready+!)	KBINTSEM	EC.IO
@T(!+/MARKSW/ ready+!)_-	MARKSEM	EC.IO
@T(!+power up+!)	ECPOWUP	EC.STATE

In addition, users may request timer-related events by supplying their own semaphores. See EC.SEQ.3

Types Provided

Type name	Where defined
access-spec	EC.SEQ.2
arraylit	EC.DATA
attribute	EC.DATA
binding	EC.DATA
bitstring	EC.DATA
boolean	EC.DATA
boolean-seq	EC.SEQ.1
call-sort	EC.SEQ.2
convar	EC.DATA
dataitem	EC.IO
defined guard	EC.SEQ.1
defitem	EC.SEQ.1
direction	EC.DATA
exclusion-relation	EC.PAR.2
guard	EC.SEQ.1
identifier-list	EC.SEQ.1
indexset	EC.DATA
integer	EC.DATA
key	EC.SEQ.1
name	EC.DATA
program name	EC.SEQ.2
real	EC.DATA
semaphore	EC.PAR.3
semaphore-attribute	EC.PAR.3
spectype	EC.DATA
statement-list	EC.SEQ.1
timeint	EC.DATA
timer	EC.SEQ.3
timer-attribute	EC.SEQ.3
typeclass	EC.DATA
type-spec	EC.SEQ.2
version	EC.DATA

Dictionary Terms

Term	Where defined
!+deadline+!	EC.PAR.1
!+destination+!	EC.DATA
!+/ENTERSW/ ready+!	EC.IO
!+failed state+!	EC.STATE
!+fall back value+!	EC.DATA
!+i/o success+!	EC.IO
!+/KBDENBL/ ready+!	EC.IO
!+/KBDINT/ ready+!	EC.IO
!+/MARKSW/ ready+!	EC.IO
!+max CPU time req+!	EC.PAR.1
!+max div result+!	EC.DATA
!+on/off+!	EC.PAR.1
!+period+!	EC.PAR.1
!+power up+!	EC.STATE
!+radix pt ident+!	EC.DATA
!+source+!	EC.DATA
!+starting event+!	EC.PAR.1
!+user threshold+!	EC.DATA
!+test result+!	EC.TEST

Undesired Events

UE name	Where defined
%already disabled%	EC.IO
%already enabled%	EC.IO
%constant destination%	EC.DATA, EC.PAR.3
%divide by zero%	EC.DATA
%guard unused%	EC.SEQ.1
%illegal array index%	EC.DATA
%illegal guard%	EC.SEQ.1
%illegal GUARDVAL use%	EC.SEQ.1
%illegal index set%	EC.DATA
%illegal INIT use%	EC.SEQ.1
%illegal PARM_GIVEN use%	EC.SEQ.2
%illegal synch%	EC.PAR.1
%inappropriate attributes%	EC.DATA
%inconsistent lengths%	EC.DATA
%inconsistent register access%	EC.DATA
%inconsistent time parms%	EC.PAR.1
%key missing%	EC.SEQ.1
%key outside IT TI%	EC.SEQ.1
%left truncation%	EC.DATA
%list mismatch%	EC.DATA
%max CPU time exceeded%	EC.PAR.1
%missed deadline%	EC.PAR.1
%name in use%	EC.PAR.2, EC.DATA
%no appropriate action%	EC.SEQ.1
%no UE correspondence%	EC.SEQ.2
%nonexistent position%	EC.DATA
%not a data item%	EC.IO
%not guard id%	EC.SEQ.1
%parm access violation%	EC.SEQ.2
%parm wrong type%	EC.SEQ.2
%range exceeded%	EC.DATA, EC.PAR.3
%read-write violation%	EC.IO
%res too fine%	EC.SEQ.3
%too few parms%	EC.SEQ.2
%too many parms%	EC.SEQ.2
%UE-handler has parameters%	EC.SEQ.2
%undeclared operand%	EC.DATA
%undeclared program%	EC.SEQ.2
%undeclared region%	EC.PAR.2
%undeclared semaphore%	EC.PAR.1, EC.PAR.3, EC.SEQ.3
%undeclared spectype%	EC.DATA
%undeclared timer%	EC.SEQ.3
%undeclared UE%	EC.SEQ.2
%undefined guard%	EC.SEQ.1

Undesired Events (continued)

UE name	Where defined
%unimplemented attribute via variables%	Appendix 4
%unimplemented binding%	Appendix 4
%unimplemented disabling%	Appendix 4
%unimplemented variable period%	Appendix 4
%unimplemented variable shift length%	Appendix 4
%unimplemented variable substring%	Appendix 4
%unknown value%	EC.DATA
%wrong init value size%	EC.DATA
%wrong init value type%	EC.DATA

System Generation Parameters

Parameter name	Data type	Where defined
#close down time#	timeint	EC.STATE
#data available (data item name)#	timeint	EC.IO
#max i/o time (data item name)#	timeint	EC.IO
#max timer error#	real	EC.SEQ.3
#min timer resolution#	timeint	EC.SEQ.3

APPENDIX 1

INTERFACE DESIGN ISSUES

EC.DATA

1. We decided to give the programmer some control over the register, so that he could take care of reducing register loads and stores by being careful with the order of operations. The alternatives we considered were notations much closer to high-level programming languages. These notations make complex expressions easier to read, but require a more sophisticated translator if we are to make efficient use of registers.

2. There is a danger with fixed point division that the results will be meaningless; this problem occurs when the numerator has more significance than the denominator. An assembly language programmer has some information that he uses to avoid this danger. The only way we can get this information is to ask the programmer to provide it, since it is dependent on the context and meaning of the division.

3. Two ways were proposed for user programs to indicate the radix of the number for a bitstring-real conversion:

   a. by giving an integer literal "i" such that the rightmost bit of the bitstring represents 2 raised to the <u>ith</u> power;

   b. by giving an integer literal "i" such that i is the number of the bit immediately to the right of the radix pt.

   Alternative 2 most closely resembles the scaling notation used in the current program, but we chose alternative 1 because most designers felt that it was easier for newcomers to understand and remember.

4. There are two main reasons for including variables whose attributes may vary:

   a) they can be reused at different points in a computation, thereby reducing the amount of space that must be reserved;

   b) they allow the same code to be used to manipulate values in widely differing ranges.

5. We require the programmer to specify a type for results stored into and retrieved from variables. We considered permitting, but not requiring, specification of the type of intermediate results and letting the Extended Computer determine the specific type when the programmer omitted the specification. We ruled out this alternative because it requires a run-time support package to keep track of the specific types of varying-type variables.

6. We considered several alternatives for providing registers:

   a) Having a common register for all type classes. This register can be very simply mapped to the accumulator.

   b) Having a separate register for each type class, implementing them with the single accumulator, and leaving the problem of interference between them up to the programmer. This was originally accepted because it is the simplest alternative that provides type checking for results in the register. However it gives away the underlying limitation, and imposes restrictions on the programmer that would not be needed if the underlying hardware had more registers or if there was multi-processor hardware.

   c) Having a separate register for each type class, implementing them with the single accumulator, and completely automating the problem of interference between the registers, freeing the programmer from any concern about it. This could be done by saving and restoring the accumulator contents whenever a different register is used. While it would be the most convenient alternative, the overhead would be prohibitive.

   d) Having a separate register for each type class, implementing them with the single accumulator, and partially automating the problem of interference between the registers. The programmer would have to indicate when he wants to reuse results in a particular register and when he does not care.

   We chose alternative (a) because it is the simplest and treats a register as a variable with varying attributes.

7. We felt it important that the EC implementation avoid saving contents of a register if they would never be needed and therefore put that burden on the programmer rather than try to do register usage analysis. We considered several ways to allow the programmer to specify whether or not the value in the register would be needed again. Among them:
   a. Associate the information with the name of the register.
   b. Associate the information with the name of the operation.

   We chose (b) because we did not want to have two names for the same object. Further, it allows us to localize the information in a place related to the operations (of which it is a property) rather than the registers.

8. An earlier version of this interface included operations such as squareroot, exponentiation, log, and root-sum-squared. We decided to move these operations to another module because they can be implemented in a machine-independent fashion. These concerns do not belong in the Extended Computer.

9. An earlier version of this module had two bitstring sizes, corresponding to halfwords and fullwords on the target computer. We then decided to have only one size because it results in a simpler data type. We finally decided to have bitstrings of any size because we noted that insisting on a fixed but unknown size made it difficult to write efficient but machine independent code. The present choice makes the interface unbiased with respect to word length and puts the burden for effective use of the actual hardware on the implementor of the EC.

10. We considered specifying bitstring sub ranges in terms of (starting point, length) instead of (starting point, ending point). One parameter fewer would be needed on bitstring compares and transfers, and we could avoid the unmatching lengths undesired event. However, we found that people working with bitstrings find it easier to work by identifying the boundary bits.

11. We considered having the EC monitor arithmetic operations for excessive loss of significance but decided that this was a programmer responsibility and could be done in a machine independent way. This eliminated the undesired event %too much lost significance%.

12. We considered relegating time to the application data type module and implementing it in terms of reals. We chose to include it in the EC because the concept of time is basic to the specification and implementation of real-time processes in the EC and because the representation should be that used in the hardware timers.

13. We considered allowing array declarations to be shared by several variables. We found this not particularly useful unless one has operations that take whole arrays as operands.

14. We decided not to allow array elements to be structures. We lose the ability to have arrays of arrays, but if this were necessary, it could be implemented in a machine independent way and could be provided by some other module.

15. We considered allowing index sets to be more general, but this seemed unnecessary even for future extensions. Such extensions could be done using the present arrays and the extension would be machine independent. We also considered restricting the lower array bound to be either 0 or 1. This seemed unnecessarily restrictive, especially as it may be desirable to select array indices at sysgen time.

16. We considered fixing the value of the array index set at declaration time, system generation time, or run time. Declaration time is too restrictive; it is sometimes useful for the array index set to be a system generation parameter. Run time fixing requires dynamic storage allocation, which is not needed or practical for avionics applications.

17. We rejected the option of operations that apply to arrays as units, e.g. multiplying arrays by scalars or arrays by arrays. Such operations depend on mathematical algorithms, rather than on characteristics of the computer and can be implemented in a machine-independent way. The present design is the simplest way to hide the hardware addressing mechanism. Extentions can be provided by user programs.

18. We considered not allowing arrays of variables whose attributes vary at run-time as it might simplify the implementation if all elements had the same attributes at all times. Although the implementation of arrays with varying attributes will probably be less efficient than arrays of fixed attribute elements, this feature is occasionally needed.

19. More than one reviewer asked if the Extended Computer shouldn't provide stacks as a builtin data structure. If we need stacks, they can be provided using the current EC facilities. The interface to those facilities (probably in the ADT module) would be carefully modelled after the EC. Should we transfer to stack machine, we could move the interface into the EC, and user programs would not have to change. This rationale also applies to floating point arithmetic, multi-dimensional arrays, array operators, etc.

20. Entity names are global in the EC. This is because that is what avionics computers provide; one can limit the scope of a name (if desired) in a machine-independent way (e.g., using naming conventions, or a pre-processor).

21. We recognize the need to represent data most efficiently for the operations in which it will be used. Since only users can determine how a datum will be used, the best the EC can do is provide a menu of representations and tell the users what each one is best and worst at doing. Hence, the "version" attribute in specific types.

EC.IO

1. This interface does not include commands for the I/O used during channel diagnostics (see EC.TEST.1). There are two reasons for this: 1) the diagnostic data items reveal secrets about how the channel works, and 2) the instruction sequences to implement test commands include timing tests as well as the normal success tests; they are dependent on the computer and are hidden in a separate diagnostics module.

2. We considered five alternatives for handling retries of unsuccesful I/O operations:

   1) having two different commands for these two cases: one that retries, either once or until it succeeds, and one that instead of retrying returns a failure indicator;
   2) having a parameter on the command specifying how often to retry, and having the command return a failure indicator;
   3) having a failure indicator, and having the user program try again if it needs to retry transmission;
   4) having a special "retry" command, with a label operand, which the user can call to have the I/O command with the specified label retried.
   5) omitting the failure indicator for the data items where it is not currently used.

   The first and fourth alternatives yield a more complicated interface than the third and provide no extra capability. The second results in extra (non-machine dependent) programs in the EC. The fifth alternative would build knowledge of the application into the EC. The third alternative relegates decisions about retrying to the user programs, and we chose this one.

3. We have considered four alternatives for handling the discrete inputs and outputs.

   Alternative 1: Treating input and output differently, allowing user programs to use a READ command to read in entire discrete input words, but providing a special WRITEBOOL command so that user programs could write individual bits appearing in the discrete output words.

   Alternative 2: Adding a READBOOL command that would read in a discrete input word, pick out the bit for a particular discrete input data item, and return it as a boolean value. Alternative 2 was rejected because not all the data items in discrete input words have boolean values. For example, /IMSMODE/ has five values, one for each switch position.

   Alternative 3: Provide the user programs with a way to specify a range of bits within both a discrete output word that they want to write out and within an input word, so that they can request individual discrete inputs in a symmetrical fashion. Alternative 3 leaves some of the responsibility for non-interference between discrete outputs to the device interface modules, since they must specify the correct ranges.

   Current: All of the above alternatives were based on the decision that the EC would sometimes identify outputs and inputs by class name rather than the individual data item name. This was done both for efficiency reasons and because it was believed that knowledge of the location of a data item within a discrete input or output word was device dependent rather than computer dependent. A much more consistent interface is achieved by always using the data item name. The EC implementer is now responsible for knowing the identity of a TC-2 I/O item, but not responsible for knowing its meaning. The efficiency problems are resolved by allowing a single command to take a list of parameters so that the EC implementation may perform operations to a single I/O word simultaneously rather than sequentially. This also eliminates special treatment of double data items.

4. We originally designed the reading of intermittent data with an access function that indicated whether or not the data were available and an undesired event if a user program tried an intermittent read operation when the data were not available. This seemed dangerous, since a slight timing difference could cause an undesired event, and the user programs could not avoid the UE. Instead, we have chosen to allow the read command at any time. If the ata are not available, the success indicator returns false. This is consistent with our general policy that it should be possible to avoid UEs by correct programming.

   Because the intermittent data is read just like any other, we decided not to have a separate command name for it.

5. We considered having serial inputs identified by class names rather than by individual data item name. Interpretation of the identification bits was considered the responsibility of the associated device interface module. We decided that identification of the data item is an EC responsibility, but interpretation of the item remains the responsibility of the DIM.

6. Note that sometimes an output should go to more than one data item. We originally handled this by letting users repeat sets of parameters to i/o commands. and saying that the order was unspecified. Since we no longer have i/o commands per se, but rather use assignment (and other bitstring) operations, we have expanded our general assignment statement so that many sources and many destinations can be given at once; the assignment happens in an unspecified order.

7. We promise that an output transmission will occur when an enabled output data item is used as a destination. We do not say when an input transmission will occur. This is because we can get away with it in the latter case, but not in the former (because an output transmission has visible effects). We hide when input takes place because someday there may be direct-memory-access input, and the computer really won't be able to control when an input item changes value.

8. We did not include the names of the data items in the main document, because we wanted to emphasize the fact that the architecture of the Extended Computer's i/o operations doesn't depend on those particular names. If the design of the Extended Computer were used with the TC-2 for some other application, the names of the data items would not be part of the technology transferred.

EC.PAR.1

1. In earlier designs of this interface, timing constraints were associated with specially designated blocks, implying that these blocks were the scheduling units. The process mechanism was unnecessarily complicated, put too many restrictions on the internal structure of processes, and gave away more information than the one here.

2. We considered having START and STOP commands so that one process can explicitly affect the ready/waiting state of another process. The problem with a STOP command is that a process cannot be safely stopped at any arbitrary point. We fixed this by adding "homing points", but specifying homing points also cluttered up the algorithm descriptions. So we dropped the idea, relying on more conventional synchronization mechanisms instead.

3. Earlier versions made an outer "do forever" loop implicit. Thus the process would execute a "INIT" block once whenever the process was started and then repeatedly execute a "FREQ" block until the process was stopped. We have decided not to include an implicit loop because we do not want to limit the internal structure of the processes. Also, the process would be easier to read if all the control was shown explicitly. Process bodies can now be specified just as subprogram bodies are, making the overall specifications of the Extended Computer simpler.

4. At one point we had intermittent processes wait for a start event and then run until a stop condition existed. We found it simpler to define a single boolean and have the process pass its start point only when the boolean was true. This eliminated the need for the event interface in the EC and eliminated ambiguous cases such as the start event ocurring when the stop condition held.

5. At one point we had a special class of processes called init processes. We recognized these as a special case of Demand processes and decided to simplify the interface by exploiting that fact. This allows some processes to be used both as init processes and under other conditions.

6. It is possible for a programmer to write a process that runs out of statements to execute. We considered three alternatives:

   a) Stating that it is an undesired event for a process to finish, i.e., making it a requirement that each process contain an infinite loop;
   b) Assuming that a completed process is in the ready state, but that it has a null statement list to execute if it becomes running;

c) Assuming that a completed process is in a waiting state, waiting for an event that will never occur.

We rejected a) because it builds too much information into the Extended Computer and it is an unnecessary restriction. We rejected b) because there is no point in having a completed process compete for a processor. Alternative 3 is a reasonable compromise for the Extended Computer interface. If it is considered undesirable to have completed processes, this should be prohibited by programming conventions.

7. We have decided not to include relative priorities for the different processes because fixed priorities do not generally work when there are real-time constraints.

8. In an earlier version, we had no distinction between periodic and demand processes because a periodic process can be viewed as one that waits for a particular stimulus, i.e., the passage of a particular amount of time. However, one of the timing parameters needed for periodic processes is not useful for demand processes. In addition, periodic processes must have restrictions on the synchronization operators within the periodic loop because the indeterminate wait associated with synchronization operators makes it difficult to prove that the loop can be scheduled regularly as required.

9. In an earlier design, we did not explicitly distinguish intermittent periodic processes. We now distinguish them in order to increase the likelihood that we can take advantage of the intermittency in the scheduling of processes. Earlier we distinguished them by calling them intermittent, now we use the presence of the optional ON_OFF to distinguish them.

10. We considered specifying periodic processes in terms of frequency rather than in terms of time intervals. Because we wanted to specify the deadline as an interval, we decided it would be more straightforward to use two intervals. These two parameters adequately constrain the variations in regularity.

11. We have an undesired event assumptions that says there won't be too many demand processes for a periodic process to miss its deadline. The assumption is worded with that orientation because it is impossible to tell how often a demand process must run.

12. We used to allow the body of a process to be any statement list. We now restrict it to a call on a previously-declared program. In this way we maintain a clear distinction between process and program, and therefore allow future extensions to include run-time creation of processes without run-time creation of programs, vice versa. The restriction does not restrict what we can do with the current version; it merely paves the way for future extensions.

EC.PAR.2

1. Regions with an exclusion relation were selected for Extended Computer synchronization primitives because
   a. they allow concurrency constraints to be expressed directly rather than as an implication of run time synchronization;
   b. they express the exclusion relationships in a form that can be interpreted efficiently by a pre-run-time scheduler;
   c. there is an algorithm for generating run-time synchronization from the exclusion relations;

   This is the simplest acceptable alternative. Rejected alternatives included:
   a. disabling interruption: once an identified section of code starts executing, it must run to completion. This alternative was rejected because it is prejudiced toward a single processor: it overly restricts the parallelism by stating that no other actions can be taken simultaneously with the code section, rather than specifying which other actions may not be taken;
   b. simple mutual exclusion: specifies all exclusion relations as equivalent, i.e., a section of code that excludes any other excludes all others. This alternative still places too many restrictions on the parallelism because many of the identified code sections need not exclude each other.
   c. named regions with mutual exclusion. Rejected because it assumes that the exclusion relation is symmetric.
   d. exclusion via synchronization primitives: using synchronization primitives such as those in EC.PAR.3 to effect mutual exclusion. Rejected because (1) synchronization primitives that are being used for other interprocess synchronization or communication purposes cannot be distinguished from those used for exclusion without additional commentary, (2) the exclusion requirements are implicit in a solution based on synchronization primitives, rather than stated explicitly as they can be with identifiable regions, and (3) the exclusion information (implying scheduling constraints) is embedded in and scattered throughout the text. These properties of the synchronism primitives make it difficult to do pre-run-time scheduling without substantial preprocessing.

2. Many useful forms of synchronization were rejected for the Extended Computer because they do not depend on the implementation of parallel process. Application-oriented synchronization operations may be developed using the exclusion relations, and semaphores (EC.PAR.3).

3. Can a region be excluded from itself? Is that useful? Yes, because in the case of non-reentrant code, this is how we will probably prevent disastrous re-invocations.

EC.PAR.3

1. We originally had more complex synchronization operators that met many immediate demands of our application. As we prefer the Extended Computer to be as application-independent as possible, we chose synchronization operations for the Extended Computer primitives that would be as simple as possible, but that could be used as building blocks for more specialized synchronization operators. For the more complex synchronization operations, see the specifications of the Application Data Type module [ADT].

   All of the following alternatives for the Extended Computer synchronization operations were rejected either because they are more complex than the operations selected or because they can be built, given the operations selected.

   a. P and V operations on semaphores;

   b. eventcounts [REED79]: Also rejected because we weren't sure we would need them;

   c. P and V supplemented by eventcounts;

   d. UP, DOWN, and PASS supplemented by event variables. A simple generalization of event variables, event-booleans can be implemented in the Application Data Type module in a machine independent way.

   e. V, DOWN, PASS, and eventcounts.

2. At one point, we provided a semaphore-to-integer conversion program. These were deleted when we could think of no reason to use it. If such a need arises, it would be a straightforward extension, allowing upward-compatability between programs written now and later.

EC.SEQ.1

1. Alternatives considered for the syntax of a guard are shown below.

   a. Boolean variables or constants only. All the boolean variables must be assigned values before the limited program is executed.
   b. Any sequence of statements assigning a boolean value to a special guarded command register.
   c. Allowing a limited program list as a guard.
   d. Allowing a program to define the value of a guard (defined guards).
   e. All of the above.

   Discussion: We chose (e). The semantics can easily be defined formally [ITTI2]. Defined guards save code by avoiding duplication of instruction lists, which would otherwise be required because of syntactic limitations.

2. We chose to have the Extended Computer provide the IT-TI construct rather than the more common IF-THEN-ELSE, CASE, and DO-WHILE constructs because IT-TI serves for all purposes. It allows some programs to be written as one loop that would otherwise require several, thereby saving variables and predicate evaluation. IT-TI has a mathematical semantics that allows systematic construction of the program's function [ITTI1].

3. Dijkstra's guarded commands are nondeterministic; of the true guards, only one is selected, but there are no rules defining which one is selected. We chose a deterministic construct because they allow simpler guards.

4. We considered providing a FOR command (FOR I = 1 to 10 DO...) but decided against it because
   a. the same purpose can be served with the IT-TI command;
   b. many special cases and questions arise with the FOR command.

5. We considered having an implicit final LP of the form (true,SKIP). We decided not to do this in order to encourage the programmer to consider every case carefully.

6. Should statement lists be allowed to contain declarations, and what is their scope? We decided that it was harmless (from this module's point of view) to allow it. The scope of all declarations is global and items must be declared before they are used, but these are issues belonging to the EC submodules that provide the declarations. (This design issue also applies to EC.SEQ.2.)

7. In an earlier version we allowed Dijkstra's cor and cand. We have eliminated them because the same effect can be obtained with the use of defined guards. For example consider

```
     +LPL+
          +LP+ +CAND+(a,b),x PL
          +LP+ c,y PL
          +LP+ true,z PL
     LPL
```

where a and b represent legal guards.

This can be written:

```
     +LPL+
          +DEF+ (m,n)
               +DLPL+
                    +DLP+ a,+DLPL+
                                   +DLP+ b,m PLD
                                   +DLP+ true,n PLD
                              LPLD
                     PLD
                    +DLP+ true,n PLD
                LPLD
          FED


          +LP+ m,x PL
          +LP+ n,+LPL+
                    +LP+ c,y PL
                    +LP+ true,z PL
                LPL
           PL
     LPL
```

As a second example consider

```
+LPL+
     +LP+ +COR+(a,b),x PL
     +LP+ c,y PL
     +LP+ true,z PL
LPL
```

This can be written:

```
+LPL+
     +DEF+ (m,n)
          +DLPL+
               +DLP+ a,m PLD
               +DLP+ b,m PLD
               +DLP+ true,n PLD
           LPLD
      FED


     +LP+ m,x PL
     +LP+ n,+LPL+
               +LP+ c,y PL
               +LP+ true,z PL
              LPL
      PL
LPL
```

EC.SEQ.2

1. Should actual and formal parameters be specified by type class, specific type name, or using type attributes such as range and resolution? We decided that type agreement should depend on the specification chosen by the programmer. Other alternatives would force us to write separate programs for each specific type or to include a parameter passing mechanism that would be more general than needed for most cases.

2. We added PARM_GIVEN because programs must be able to tell if an optional parameter was supplied or not. We thought about making it a built-in value that any type of variable could take on; then programmers could ask, e.g., if p1=PARM_GIVEN. However, because output parameters can be optional, we didn't want programmers checking their "value".

3. Programmers need not supply trailing commas when optional parameters at the end of a parameter list are omitted. That is, instead of +pgm1+(a,b,,,) one may write +pgm1+(a,b). Besides the obvious convenience, this will allow us to add optional parameters to the end of any access program parameter list, yet not force all calls on that program to change.

4. We added the feature of ranking programs' access speed because the current computer has the capability of doing fast subroutine linkages in certain areas of memory. Because a replacement machine may not have such a capability, we made the relationship "not-slower-than", which we can trivially implement by doing nothing. We make no firm promise about the ordering, however, because we recognize that we cannot make access to a subroutine not-slower-than access to an expanded macro that simply lives in-line.

EC.SEQ.3

1. In earlier versions, we had clocks and timers; clocks counted up and timers counted down. They were completely distinct from timeint entities; they were declared separately and had their own set of operations. We removed the distinction as the interface grew and grew, and we realized that it would be both useful and consistent to let a clock/timer do most anything that a timeint can do.

2. We considered providing only clocks or only timers (in the sense of issue #1). Clocks are useful for measuring elapsed time; timers are useful for detecting the end of a previously specified time interval. We wanted the capabili'ies of both because otherwise user programs would have to use one to simulate the other. This would lead to inefficiency and possible duplicate efforts especially on a computer that provided both.

3. We considered having this module offer a special "waittime" command, instead of using the general semaphore mechanism. There seems to be no advantage in using a special mechanism for timed events.

3. We considered treating the following actions as errors:
   - starting a running timer,
   - setting a running timer,
   - stopping a non-running timer,
   - reading a non-running timer,
   - stopping a timer that has run down,
   - reading a timer that has run down,

   but these actions are not necessarily senseless.

4. We considered having a timer signal a UE if it runs past its capacity. To have it start over seems the most useful. Further, a timer might run past its usual limit, for no fault of the software. In contrast, setting timer with too large a value is a clear software error. Therefore we made exceeding the maximum capacity an undesired event in a set operation.

5. In an earlier design, there was a single maximum capacity for all clocks and a single maximum for all timers. It was pointed out that clocks and timers are used for very different purposes, some for measuring very small changes over a small period of time, and some for keeping track of a long period of time, with less concern for small changes. In order to achieve this flexibility without undue use of resources, we decided to allow programmers to specify capacity and minimum measurements for individual timers and clocks.

6. In an earlier version, clocks could only be set to zero, but this seems unnecessarily restrictive. Dwight Hill: "I believe we may need a +SET_CLOCK+ for clock corrections or for time-of-day clocks." The restriction went away when we merged timers and timeints.

EC.STATE

1. The following transitions are not included in this interface for the following reasons:

off to failed:	not relevant to user programs;
failed to off:	user programs cannot respond to anything
operating to off:	when the computer is off;

   Note that failed to operating does not occur with the current computer; it must be cycled through "off" to get back to operating from failed. However, future computers may make this transition possible (perhaps by re-booting), and so this transition is subsumed by the definition of power up.

2. There may not always be a grace period after @T(!+failed state+!). Two alternatives were considered: to leave out the grace period altogether, or to include it as a system-generation parameter. We selected the latter to allow for future use of an improved computer.

3. How do we distinguish between malfunctions that user programs must detect and handle (possibly by calling +S_FAIL_STATE+) and malfunctions that are detected inside the Extended Computer? Malfunctions are detected by this module if they are reported by the computer without software action; for example, malfunctions signalled by interrupts. Whenever a malfunction is detected because of an action dictated by the requirements, such as a diagnostic test, detection is left to a user program. The malfunctions described in EC.TEST belong to the latter category; all others, the former.

4. Future technology may make our three-state model appear oversimplified, because a system may have degraded states: that is, states without the full capability of "operating", yet not dead in the water like "failed". A degraded state may occur in a single-processor system, or in a multi-processor system where one or more processors have ceased to operate. It is important that acquiring this capability results in adding to (not revising) the present specification. We cannot add a degraded state now, because we cannot implement and programs depending on it would be not be correct. However, we can plan for the addition by assuming that there are "<u>at least</u> three states", etc.

5. Which module is responsible for the close-down procedures? We decided that any shutdown action that is required for every computer failure and is computer-dependent should be done by this module. If the action is device-dependent, such as setting the bomb-release output to a safe value, it should be done by the device interface module.

EC.TEST

1. The signal converter is tested by sending particular values to it and then reading back the results of the internal signal converter manipulation on the values. The proper relationship between the values sent out and the value read in can be characterized by a set of equations. The design issue is how much of the knowledge should be hidden within this module: both the equations and the choice of test values, just the equations, or neither. The equations are based on the behavior of the channel, and therefore belong within this module. The choice of values could be considered part of the software requirements; they affect the displays seen by the pilot, and are documented in section 4 of the requirements. However, the choice of these particular values is partly influenced by hardware characteristics. Further, if they are not hidden, the interface to this module becomes much more complex. We have chosen to hide all of the information even though it means hiding some details about the required functions in this interface. We assume that the test values are likely to change with the hardware and not for any other reason.

2. We decided to hold user programs responsible for avoiding interference between the diagnostics and the regular commands rather than build monitors into the I/O commands and diagnostics. The diagnostics are not expected to be run when the software is doing anything else. Monitors impose a run-time cost in the regular commands.

3. We considered dividing memory into banks that would be tested separately, allowing partial rather than complete shutdown. We decided not to do so at this time because the system lacks the ability to exploit it and we could do so easily in the future.

4. A previous design implied that invoking the access program associated with a test actually started the test. Because future computers may have tests ongoing, or running in the background, we changed our design to indicate that invoking a program merely returns the most recent result of that test. If a future computer is <u>required</u> to start a test at a certain time, we can add start-test commands later. Returning the value may take a substantial amount of time in some cases. The major change this caused was in the case of the memory test. Before, there was a command to start the test, an event signalled when it was done, and a program to retrieve the result. The motive was that the invoking program would want to do something else while waiting for the test to be completed. However, some program would have to wait idly for the event to occur anyway, and so we lose nothing by letting the memory test program just take a long time to return. We gain a uniform interface, with no special cases.

APPENDIX 2

IMPLEMENTATION NOTES

EC.DATA: None.

EC.IO

1. The part of the I/O submodule that handles the relation between data item names and TC-2 instruction sequences should be a sysgen time program and should be table driven. It should be organized into submodules in accordance with the structure of the Device Interface Module, because changes are likely to be concentrated on individual devices.

EC.PAR.1: None.

EC.PAR.2: None.

EC.PAR.3: None.

EC.SEQ.1: None.

EC.SEQ.2

1. This module does not determine where programs are located in memory. It uses programs in the memory allocator module to request space.

2. This module uses the System Generation module to do assembly-time parameter type checking.

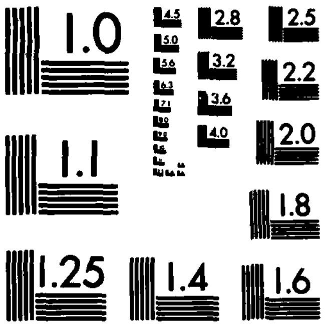
EC.SEQ.3: None.

EC.STATE: None.

EC.TEST: None.

APPENDIX 3

BASIC ASSUMPTIONS



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

EC.DATA

1. The Extended Computer can provide one private variable per process (the register) that can store values of any type. Access to a register will usually be quicker than access to other variables.

2. The attributes of a value will be known whenever a variable is used as a source or a destination. If the attributes specified for the variable when it is used as a source are not the same as were specified when its value was determined, the result may be any value.

3. The Extended Computer can store numeric quantities with any desired range and resolution. It can be expected that (a) variables with a small range-to-resolution ratio will require less actual memory space than variables with a large range-to-resolution ratio, and (b) that operations on such variables will be faster than operations on variables with a larger range-to-resolution ratio.

4. Range and resolution are adequate characterizations of a numeric variable; i.e., the needs of an application programmer can be adequately expressed by a lower bound, upper bound and guaranteed resolution.

5. The Extended Computer can store bitstring quantities of any desired length. Longer bitstring entities may require more storage than shorter ones. Operations on longer bitstring entities may require more computer time than operations on shorter ones.

6. Whenever a numeric value is stored into a variable with a resolution different from the source, the value stored should always be the closest value that can be represented in the destination. The programmer need not specify the conversions to be made; the best choice can be made by the EC implementation.

7. There is no need for operations that allow a bitstring value of one length to be assigned to a bitstring variable with a different length.

8. The operations needed for calculating new numeric values are:

   addition, multiplication, division, subtraction, absolute value, complement and conversions.

9. Division may result in a loss of all significance. This danger cannot be hidden entirely from the programmers, since they may have information that can be used to choose safe, efficient algorithms. The following division options are sufficient:

   a. The quickest division can be performed if the programmer provides an upper bound for the result. The better the bound, the more significance is preserved. If the bound is too low, all significance may be lost.

   b. A slower algorithm can be used if the programmer cannot provide an upper bound.

   c. If the programmer cannot provide an estimate of the maximum result and prefers to avoid the expense of the slower algorithm, the Extended Computer can determine whether or not division can be safely performed. The EC can return the sign of the quotient even when the operation cannot be safely performed.

10. The rules in Table EC.DATA.d can be implemented.

11. Whenever the program compares two numeric operands for equality, programmers need to define a threshold, such that if the difference between two numbers is less than or equal to the threshold, the numbers are considered equal. If no threshold is specified, two numeric operands can be considered equal if the difference between them is smaller than one-half of the larger of the two guaranteed resolutions associated with the operands.

12. It is acceptable for the results of an operation to have a larger resolution than the resolution of the destination. The approximations needed to store the result can be assumed to be acceptable for the application.

13. Only four kinds of entities are needed: variables, which can be changed at any time; ascons and literals, which can be changed by reassembling the program; and loadcons, which can be changed when the program is first loaded into the computer but not while it is running.

14. The following operations are sufficient for efficiently producing new bitstring values from existing bitstring values:

    a. AND, OR, NAND, NOT, MINUS, and XOR, defined in the usual way, operating on corresponding bits in two operands of equal length;

    b. SHIFT operation: A bitstring is shifted either right or left a specified number of bits with zeros shifted into positions vacated by the shift;

    c. REPLACE operation: A portion of a bitstring is replaced by the value found in an equal-length portion of another bitsring;

    d. CAT operation: A bitstring is formed by concatenating two previously existing bitstrings.

15. If the result of converting a real to a bitstring has more bits than the bitstring operand, the bits to the right of the rightmost bit of the destination bitstring may be ignored.

16. Arrays with dimensions that vary at run time are not needed in avionics applications.

17. Avionics applications do need arrays in which the type class is real, and the elements are variables with attributes that may vary independently of the attributes of other elements of the same array.

18. Arrays in which the indices are not a contiguous subset of the integers are not needed in avionics applications.

19. Avionics applications need to take advantage of any capability that the computer has to allow faster memory access to certain data. The Extended Computer can implement a "not-slower-than" relation for any two declared entities x and y, so that x will be accessed no slower than y. User programs can determine desired rankings at system generation time; it is not necessary to change the rankings at run-time.

EC.IO

1. The only information needed by user programs to identify inputs or outputs is the data item name given in the requirements document [REQ]. It is possible to characterize all transmissions between the Extended Computer and its associated hardware as either input or output.

2. Input data items and output data items are bitstring entities. Some can only be used as a source in a statement (read-only); some can only be used as a destination in a statement (write-only); some can be used as either (read-write). No input data item is write-only. No output data item is read-only.

3. It is possible to turn off (disable) input/output transmissions. A disabled data item has no effect on and is not affected by the external environment.

4. No application program will need the identity code and subitem identifiers in Serial Input Register Data (see [REQ]).

5. It is possible for the software to determine the success of I/O operations. (Of course, this assumption is obviously false if we consider hardware failures. However the correctness of our software is contingent on that assumption.) An unsuccessful operation may not change the value of the associated data item.

6. Some input data items are only available intermittently and the EC can notify user programs when such data become available. The data are expected to remain available for an amount of time that (for each item) can be determined at system-generation time. Other data items can be expected to be available at any time.

7. Each i/o operation can be guaranteed to complete within a fixed period of time. This worst-case timing requirement varies among data items; the time associated with each data item can be determined at system-generation time.

EC.PAR.1

1. Processes (executions of programs) may execute in parallel with no restrictions on their relative speeds, except where they are explicitly synchronized with each other (see EC.PAR.2).

2. The number of processes need not vary at run-time. It may be set at system generation time.

3. All demand processes can start when the system is turned on (i.e., when @T(!+power up+!) occurs); some will perform initialization routines; the remaining demand processes will wait for a semaphore to become nonnegative.

4. The process mechanism will be able to detect the event @T(!+power up+!).

5. Processes are not called as subroutines by other programs and do not return control to other programs.

6. We need only distinguish two process states: <u>active</u> or <u>suspended</u>. An active process can progress. A suspended process is ineligible to progress (continue execution).

7. The state of a process changes between active and suspended only when it uses the process synchronization mechanisms described in sections EC.PAR.2 and EC.PAR.3 or when it has executed the last statement (END) in its body.

8. All processes are either periodic or demand and exist throughout the life of the system;

   The bodies of periodic processes are to be executed at regular intervals (their period). The period of a process may change during system execution. A periodic process may be suspended when a specified boolean variable is false and start again when it is true.

   Demand processes wait for a semaphore to be nonnegative. They should be executed each time the semaphore is incremented. They will decrement the semaphore once per execution.

9. Demand processes can be adequately characterized by specifying the values of two timing parameters: maximum CPU time requirement and deadline for completion.

10. Periodic processes are adequately characterized by three timing parameters: maximum CPU time requirement, deadline, and period.

EC.PAR.2

1. User programs may contain contiguous sections or <u>regions</u> of run-time-executable statements that may not be executed concurrently. These concurrency constraints can be expressed in terms of an exclusion relation on the regions, i.e., where region 1 <u>excludes</u> region 2 if region 2 may not start while region 1 is executing.

2. Regions may overlap other regions or be embedded in other regions.

EC.PAR.3

1. There are two process states relative to synchronization: active (which includes processes that are running and processes that are ready) and suspended (ineligible to make progress). The active processes are the only ones eligible for execution.

2. The only operations on semaphores that need to be executed in a way that guarantees non-interference with other operations on semaphores are the following:
   a. An operation that does not affect the counter value of the semaphore, but may put the process in the waiting state.
   b. An operation to decrement the semaphore counter without any effect on the state of the process that executes it.
   c. An operation to increment the semaphore counter that may put other processes in the active state.

EC.SEQ.1

1. The only sequence control constructs needed are those that (a) limit the states under which a statement list will be executed (and alter the state before executing the statement list); (b) select among alternative statement lists, and (c) iterate if one of a specified set of alternatives is chosen. There is no need for a go to.

2. Statements appearing within the sequence control constructs have exactly the same semantics as they do anywhere else.

EC.SEQ.2

1. It is necessary to have parameterized procedures. These parameters will be named by the programmer and are part of the specification of the program.

2. The implementor of a program can specify the number, position, and type of the program's parameters. (If a formal parameter is an array, its specification includes the type of the array elements.)

3. Some programs should be invoked faster than others. Such a relation will not depend on when the programs are invoked; the relative ranking can be determined at system generation time.

4. If a program will be reentered while already in use by another process, it is the responsibility of the programmer to make sure that local storage is saved and restored as needed. EC programs are not automatically provided with new storage when they are reentered.

5. There is no need for a mechanism to allow programs to cause the calling program to resume execution anywhere else than immediately after the call.

6. There is no need for programs with more than one entry point.

7. The identity of a data entity that is passed to a program as an actual parameter will not be changed while the program is executing. For example, when an array element is passed as an actual parameter to a program, if that program alters the value of the variables that determined the index, the results will be undefined.

8. Parameters always fall into one of three classes: I, O, and IO as defined in EC.SEQ.2.4.

9. It is necessary to provide facilities for recovery if a programming error is detected by a program during execution. It is up to the author of a called program to determine what programming errors his program can detect; it is up to the caller of a program to determine the action that should be taken if one of those errors occurs. It is not necessary to pass parameters to the recovery program.

10. A program must be able to detect whether or not optional parameters were supplied in an invocation of that program.

EC.SEQ.3

1. Avionics programs need timers that keep track of elapsed time, and that may signal when a given time interval has elapsed. They need to be able to set a timer to a starting value, start it, stop it, and read it whether it is running or not.

2. The maximum timing capacity of a clock or a timer can be determined at system generation time.

3. If a timer runs beyond a limit specified at run time, it should either halt or start over. Sometimes it should signal that a range limit has been reached.

5 The worst acceptable error rate for all timers can be determined by users at system generation time. This error can be specified as a fraction of the running time.

6. Any number of timers can be implemented, provided that the number is known at system-generation time. There is no need to create or delete timers at run time.

EC.STATE

1. The Extended Computer has at least three states: off, operating, and failed. Only the following transitions between states affect user programs:

   - from off to operating
   - from operating to failed.

2. User programs cannot cause the transition into the operating state.

3. A transition from operating to failed can either be caused by user programs or occur when malfunctions internal to the Extended Computer are detected. These internal malfunctions are other than those described in EC.TEST. It should be assumed that after this transition occurs, user programs will have at least a short interval to execute shut-down sequences before the computer stops operating. The minimum length of the interval before shut-down can be determined at system-generation time.

4. Any actions that must be taken when a computer failure occurs are independent of the state of the user programs, and can be built into the EC.

EC.TEST

1. Each channel diagnostic program may interfere with a specified subset of the input/output commands. They will not interfere with any other commands.

2. Use of either the discrete diagnostics or the accelerometer-torque diagnostics may cause the IMS to lose its alignment and velocities (i.e., have the same effect as disabling the IMS temporarily).

3. The following aspects of the input/output can be tested independently:

   the AC aspects of the signal converter channel,
   the DC aspects of the signal converter channel,
   the cycle steal channel A and serial input channel 1,
   the cycle steal channel B and serial input channel 2,
   discrete input word 1 and discrete output word 1,
   discrete input word 2 and discrete output word 2,
   discrete input word 3 and discrete output word 3,
   the IMS gyro torque registers and the accelerometer accumulators.

4. A memory diagnostic program can check whether portions of memory are reliable. This program does not interfere with other programs. The test may take a substantial amount of time to complete.

5. There are diagnostic programs that can test the hardware timers and the interrupt mechanism separately, but may interfere with proper execution of other programs.

APPENDIX 4

UNIMPLEMENTED EXTENDED COMPUTER FACILTIES

Not all of the capabilities described in this document have been provided in the current version of the Extended Computer. A few facilities, which are not currently needed by the application program, have not been implmented. An attempt to use an absent facility will result in an undesired event in the development version. The unimplemented features are described below.

FEATURE:	Periodic processes with periods that vary at run-time
WHERE DESCRIBED:	EC.PAR.1
UNDESIRED EVENT:	%unimplemented variable period%
CURRENT USE:	The !+period+! parameter in the ++P_PROCESS++ must be given as a constant or a literal.

FEATURE:	Ability to enable/disable all data items
WHERE DESCRIBED:	EC.IO
UNDESIRED EVENT:	%unimplemented disabling%
CURRENT USE:	Only the following data items may be disabled; attempting to +DISABLE+ or +ENABLE+ any other is prohibited.

//ASAZ//	//ASEL//	//ASLAZ//
//ASLEL//	//ASLCOS//	//ASLSIN//
//AZRING//	//BAROHUD//	//CURAZCOS//
//CURAZSIN//	//CURPOS//	//DESTPNT//
//FLTDIRAZ//	//FPMAZ//	//FPMEL//
//HUDAS//	//HUDASL//	//HUDFPM//
//HUDPUC//	//HUDSCUE//	//HUDVEL//
//HUDWARN//	//LSOLCUAZ//	//LSOLCUEL//
//MAGHDGH//	//MAPOR//	//PTCHANG//
//PUACAZ//	//PUACEL//	//ROLLCOSH//
//ROLLSINH//	//USOLCUAZ//	//USOLCUEL//
//VERTVEL//	//VTVELAC//	//XCOMMC//
//XCOMMF//	//YCOMM//	

FEATURE:	Bitstrings/timeints with attributes that can vary at run-time
WHERE DESCRIBED:	EC.DATA
UNDESIRED EVENT:	%unimplemented binding%
CURRENT USE:	In the ++DCL_TYPE++ program, users may not declare the binding of bitstring or timeint specific types to be VARY. In the ++DCL_ENTITY++ and ++DCL_ARRAY++ programs, users may not provide an initial attribute.

FEATURE:	Timers with attributes that can vary at run-time
WHERE DESCRIBED:	EC.SEQ.3
UNDESIRED EVENT:	%unimplemented binding%
CURRENT USE:	In the ++DCL_TYPE++ program for timers, users may not declare the binding of timers to be VARY. In the ++DCL_ENTITY++ program for timers, users may not supply an initial attribute.

FEATURE:	Semaphores with attributes that can vary at run-time
WHERE DESCRIBED:	EC.PAR.3
UNDESIRED EVENT:	%unimplemented binding%
CURRENT USE:	In the ++DCL_TYPE++ program for semaphores, users may not declare the binding of semaphores to be VARY. In the ++DCL_ENTITY++ program for semaphores, users may not supply an initial attribute.

FEATURE:	Undesired events in the production EC
WHERE DESCRIBED:	Throughout
UNDESIRED EVENT:	none
CURRENT USE:	In the production version of the Extended Computer, no undesired events will be checked for; no undesired event handling programs will be assembled or executed. It will be assumed that user programs will invoke the EC facilities correctly.

FEATURE:	Specifying substrings of bitstrings with variables
WHERE DESCRIBED:	EC.DATA.2.7.2
UNDESIRED EVENT:	%unimplemented variable substring%
CURRENT USE:	In the bitstring +REPLC+ program, p2, p3, and p4 must be given by literals or constants.

FEATURE:	Specifying the length of a bitstring shift with a variable
WHERE DESCRIBED:	EC.DATA.2.7.2
UNDESIRED EVENT:	%unimplemented variable shift length%
CURRENT USE:	In the +SHIFT+ program, p2 must be given by a literal or constant.

FEATURE:	Using variables to specify attributes of a specific type, or of a variable or array with varying attributes
WHERE DESCRIBED:	EC.DATA.4, EC.PAR.3.4, EC.SEQ.3.4
UNDESIRED EVENT:	%unimplemented attribute via variables%
CURRENT USE:	To specify an attribute (as defined in EC.DATA.4), a timer-attribute (as defined in EC.SEQ.3.4), or a semaphore-attribute (as defined in EC.PAR.3.4), literals or constants must be used.

APPENDIX 5

INPUT/OUTPUT DATA ITEM NAMES

The following table lists all data items known to the Extended Computer, and tells whether each one is read-only (R), write-only (W), or read-write (RW).

INPUT DATA ITEMS

Data item name	R or RW
/ACAIRB/	R
/ADCFAIL/	R
/AOA/	R
/ANTGOOD/	R
/ARPINT/	R
/ARPPAIRS/	R
/ARPQUANT/	R
/BAROADC/	R
/BRGSTA/	R
/BMBDRAG/	R
/DIMWC/	R
/DRFTANG/	R
/DGNDSP/	R
/ANTGOOD/	R
/ELECGOOD/	R
/DRSFUN/	R
/DRSMEM/	R
/DRSREL/	R
/ENTERSW/	R
/FLYTOTW/	R
/FLYTOTOG/	R
/GUNSSEL/	R
/HUDREL/	R
/IMSAUTOC/	R
/IMSMODE/	R
/IMSREDY/	R
/IMSREL/	R
/KBDENBL/	R
/KBDINT/	R
/LOCKEDON/	R
/MACH/	R

OUTPUT DATA ITEMS

Data item name	W or RW
//ASAZ//	RW
//ASEL//	RW
//ANTSLAVE//	W
//ASLAZ//	RW
//ASLEL//	RW
//ASLCOS//	RW
//ASLSIN//	RW
//AUTOCAL//	W
//AZRING//	RW
//BAROHUD//	RW
//BRGDEST//	W
//BMBREL//	W
//BMBTON//	W
//COMPCTR//	W
//COMPFAIL//	W
//CURAZCOS//	RW
//CURAZSIN//	RW
//CURENABL//	W
//CURPOS//	RW
//DESTPNT//	RW
//ENTLIT//	W
//FIRRDY//	W
//FLTDIRAZ//	RW
//FPANGL//	W
//FPMAZ//	RW
//FPMEL//	RW
//FLTREC//	W
//GNDTRK//	W
//GNDTRVEL//	W
//HUDAS//	RW
//HUDASL//	RW

INPUT DATA ITEMS

Data item name	R or RW
/MAGHCOS/	R
/MAGHSIN/	R
/MARKSW/	R
/MA/	R
/MFSW/	R
/MODEROT/	R
/MULTRACK/	R
/PNLTEST/	R
/PCHCOS/	R
/PCHSIN/	R
/PMDCTR/	R
/PMHOLD/	R
/PMNORUP/	R
/PMSCAL/	R
/PRESPOS/	R
/RADALT/	R
/RNGSTA/	R
/RE/	R
/ROLLCOSI/	R
/ROLLSINI/	R
/SINEVEL/	R
/SINHDG/	R
/SINLAT/	R
/SINLONG/	R
/SINNVEL/	R
/SINPTH/	R
/SINROL/	R
/SLTRNG/	R
/SLEWRL/	R
/SLEWUD/	R
/STA2RDY	R
/STA3RDY/	R
/STA6RDY/	R
/STA7RDY/	R
/STA8RDY/	R
/TD/	R
/TAS/	R
/THDGCOS/	R
/THDGSIN/	R
/UPDATTW/	R
/WAYLON/	R
/WAYLAT/	R

OUTPUT DATA ITEMS

Data item name	W or RW
//HUDFPM//	RW
//HUDPUC//	RW
//HUDSCUE//	RW
//HUDVEL//	RW
//HUDWARN//	RW
//IMSNA//	W
//IMSSCAL//	W
//KELIT//	W
//LATGT70//	W
//LFTDIG//	W
//LSOLCUAZ//	RW
//LSOLCUEL//	RW
//LWDIG1//	W
//LWDIG2//	W
//LWDIG3//	W
//LWDIG4//	W
//LWDIG5//	W
//LWDIG6//	W
//LWDIG7//	W
//LLITDEC//	W
//LLITE//	W
//LLIT322//	W
//LLITW//	W
//MAGHDGH//	RW
//MAPOR//	RW
//MARKWIN//	W
//PTCHANG//	RW
//PUACAZ//	RW
//PUACEL//	RW
//RNGUNIT//	W
//RNGTEN//	W
//RNGHND//	W
//RNGUNIT/	W
//ROLLCOSH//	RW
//ROLLSINH//	RW
//STEERAZ//	W
//STEEREL//	W
//STERROR//	W
//TSTADCFLR//	W
//USOLCUAZ//	RW
//USOLCUEL//	RW
//ULITN//	W

INPUT DATA ITEMS

Data item name	R or RW
/WAYNUM1/	R
/WAYNUM2/	R
/WEAPTYP/	R
/XVELTEST/	R
/XVEL/	R
/XGYCNT/	R
/YVELTEST/	R
/YVEL/	R
/YGYCNT/	R
/ZVELTEST/	R
/ZVEL/	R
/ZGYCNT/	R

OUTPUT DATA ITEMS

Data item name	W or RW
//ULIT222//	W
//ULIT321	W
//ULITS//	W
//UWDIG1//	W
//UWDIG2//	W
//UWDIG3//	W
//UWDIG4//	W
//UWDIG5//	W
//UWDIG6//	W
//VERTVEL//	RW
//VTVELAC//	RW
//XGYCOM//	W
//XTORTEST//	W
//XCOMMC//	RW
//XCOMMF//	RW
//XSLEW//	W
//XSLSEN//	W
//YGYCOM//	W
//YTORTEST//	W
//YCOMM//	RW
//YSLEW//	W
//YSLSEN//	W
//ZGYCOM//	W
//ZTORTEST//	W
//ZSLEW//	W
//ZSLSEN//	W

The following data items have events (signalled by incrementing a semaphore) associated with them:

<u>Event</u>	<u>Semaphore</u>
@T(!+/ENTERSW/ ready+!)	ENTSWSEM
@T(!+/KBDENBL/ ready+!)	ENBLSEM
@T(!+/KBDINT/ ready+!)	KBINTSEM
@T(!+/MARKSW/ ready+!)	MARKSEM

APPENDIX 6

DATA REPRESENTATION CATALOGUE

For some data types, the Extended Computer is capable of providing more than one kind of representation. The version has no effect on the outcome of an EC operation, but some versions allow some operations to be performed more quickly than other versions.

The following table lists the provided version names for each EC data type which has more than one version. When declaring a specific type, users may request a particular version by using these names.

TABLE TBD

REFERENCES

[ADT] Clements, P., Interface Specifications for the A-7E Application Data Type Module; NRL Memorandum Report in preparation, April 1982.

[APC] Faulk, S.; "Pseudo-Code Language for the A-7E OFP", internal memorandum, April 1982

[BELP73] Belpaire and Wilmotte, "A Semantic Approach to the Theory of Parallel Processes"; in International Computing Symposium 1973.

[DIJK68] Dijkstra, E.; "Cooperating Sequential Processes", in Programming Languages, ed. F. Genuys; Academic Press, 1968. pp. 43-112.

[DIJK77] Dijkstra, E.W.; A Discipline of Programming; Prentice Hall, c1976.

[DIM] Parker, R.A., Heninger, K.L., Parnas, D.L., Shore, J.E.; Abstract Interface Specifications for the A-7E Device Interface Module, NRL Memorandum Report 4385, November, 1980.

[FIXPT] Heninger, K.; "Prototype Implementation of the Fixed Point Data Type Module"; NRL Technical Memorandum 7503-210:KH:kh, dtd 31 July 1979.

[ITTI1] Parnas, D. L.; An Alternative Control Structure and Its Formal Definition, Technical Report FSD-81-0012, Federal Systems Division, IBM Corporation, Bethesda, MD., 1981; accepted for publication in Comm. of the ACM.

[ITTI2] Parnas, D. L.; Understanding Programs, Technical Report, Federal Systems Division, IBM Corporation, Bethesda, MD, in preparation.

[REAL] Heninger, K.; "Prototype Implementation of the Real Data Type Module"; NRL Technical Memorandum 7503-209:KH:kh, dtd 27 July 1979.

[REED79] Reed, D.P. and R.K. Kanodia; "Synchronization with Eventcounts and Comm. of the ACM, v. 22, no. 2 pp. 115 (1979).

[REQ] Heninger, K.L., Kallander, J.W., Parnas, D.L., Shore, J.E.; Software Requirements for the A-7E Aircraft; NRL Memorandum Report 3876; Nov 1978.

[SO] Software Cost Reduction project, Standard Organization for Describing Abstract Interface Specifications; NRL Memorandum Report in preparation. Until publication, readers should consult the Standard Organization chapter of [DIM] instead.

[TRACE] Parnas, "Trace Specifications for D-Operations", NRL Technical Memorandum 7590-000:DP, to be published.

[WUER76] Wuerges, H. and Parnas D.L.; "Response to Undesired Events in Software Systems"; Proc. Second Int. Conf. Software Eng., pp. 437-446; 1976.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the hard work and careful reviews provided by the following people:

Naval Weapons Center, China Lake, CA:
- Jack Basden
- Richard Fryer
- Sandra Fryer
- Dawn Janney
- Ray Martinusen
- Jo Miller
- Lee Thomson
- Robert Westbrook
- Richard Wolff
- Janice Zenor

Vought Corporation, Dallas, TX:
- Glenn Cooper
- Dwight Hill

USAF A-7D/K OFP Detachment, Tucson, AZ:
- Mark Jacobson
- Richard Breisch

Bell Telephone Laboratories, Columbus, OH:
- Don Utter

Grumman Aerospace Corp., Bethpage, NY:
- Stephanie White